2012-03

Experimental Tracking of Aerial Targets Using the Microflown Sensor

Ng, Chee Wee

Monterey, California. Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXPERIMENTAL TRACKING OF AERIAL TARGETS USING THE MICROFLOWN SENSOR

by

Chee Wee Ng

March 2012

Thesis Co Advisors:	Daphne Kapolka Kevin B. Smith

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2012	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Experimental Tracking of Aerial Targets Using the Microflown Sensor			**5. FUNDING NUMBERS**
6. AUTHOR(S) Chee Wee Ng			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ___N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Determining target bearing based on a passive acoustic signal typically relies on beamforming the signals from an array of sound pressure sensors. A major drawback, however, is the proportional increase in array aperture when dealing with low frequencies, such as the lengthy towed arrays used for anti-submarine warfare. This thesis demonstrates the use of a single acoustic vector sensor (Microflown Ultimate Sound Probe (USP)) to derive the target bearing by processing both the pressure as well as particle velocity information of an acoustic wave. Field experiments were set up to track commercial aircraft during their final approach before landing. Despite healthy signal-to-noise (SNR) ratios, significant challenges were faced in accurate real-time tracking. Post-processing frequently achieved better results, but required the beamformer to process a broader range of frequencies (typically 300-1000 Hz), instead of focusing on narrowband energy peaks. This was attributed to the effects of noise and bottom reflections (mainly from the concrete ground), as implied by the distinct Lloyd's mirror patterns in the spectrograms. Notwithstanding, additional information such as target altitude and horizontal distance at the closest point of approach (CPA) could be determined from analyzing these patterns.

14. SUBJECT TERMS Acoustic Vector Sensor, Particle Velocity, Beamforming, Noise, Microflown, Real-time Tracking, Aircraft			**15. NUMBER OF PAGES** 95
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

EXPERIMENTAL TRACKING OF AERIAL TARGETS USING THE MICROFLOWN SENSOR

Chee Wee Ng
Major, Singapore Navy
B.Eng., National University of Singapore, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING ACOUSTICS

from the

NAVAL POSTGRADUATE SCHOOL
March 2012

Author:	Chee Wee Ng
Approved by:	Daphne Kapolka Thesis Co-Advisor
	Kevin B. Smith Thesis Co-Advisor
	Daphne Kapolka Chair, Engineering Acoustics Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Determining target bearing based on a passive acoustic signal typically relies on beamforming the signals from an array of sound pressure sensors. A major drawback, however, is the proportional increase in array aperture when dealing with low frequencies, such as the lengthy towed arrays used for anti-submarine warfare. This thesis demonstrates the use of a single acoustic vector sensor (Microflown Ultimate Sound Probe (USP)) to derive the target bearing by processing both the pressure as well as particle velocity information of an acoustic wave.

Field experiments were set up to track commercial aircraft during their final approach before landing. Despite healthy signal-to-noise (SNR) ratios, significant challenges were faced in accurate real-time tracking. Post-processing frequently achieved better results, but required the beamformer to process a broader range of frequencies (typically 300-1000 Hz), instead of focusing on narrowband energy peaks. This was attributed to the effects of noise and bottom reflections (mainly from the concrete ground), as implied by the distinct Lloyd’s mirror patterns in the spectrograms. Notwithstanding, additional information such as target altitude and horizontal distance at the closest point of approach (CPA) could be determined from analyzing these patterns.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I wish to thank the wonderful faculty, laboratory support staff, and fellow students of the Undersea Warfare community at NPS, without which this thesis would not have been possible, and certainly not nearly as enjoyable. Many thanks to Jay Adeff and Sam Barone for your wonderful support and friendship.

My greatest gratitude to my thesis advisor, Daphne Kapolka. Your instructions are amongst the most engaging and memorable in my time at NPS. Your guidance and patience were most appreciated in the course of my research. You are a wonderful teacher, and most certainly now a dear friend.

Lastly, I like to thank my dear wife Cheryl, for your support all this while and for taking care of matters at home so I can concentrate on work at school. To my boys Ryan and Jake, nothing beats coming home every day to your lovely smiles and the ever-so-often carpet soccer game. You remind me to always have fun in life, no matter what.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The use of acoustics in target detection is of interest to many fields and has conventionally been achieved via phase beamforming spatially distributed sensors. The sensors first capture the pressure signals of the incoming acoustic waves. The signals are then combined in a beamformer to determine the direction of arrival (DOA). Such methods, however, have a major drawback. In order to achieve good bearing resolution, the aperture of an array must be large compared with the wavelength. Furthermore, a single linear array has complete azimuthal symmetry making it impossible to distinguish which side of the array the target is on. In low-frequency applications, they usually result in large physical footprints with consequently high cost, limited mobility, and counter-detection issues.

Acoustic vector sensors, on the other hand, have the potential of achieving better DOA estimates with smaller arrays. Each acoustic wave gives rise to **pressure** as well as **particle velocity** changes in the transmission fluid. Knowledge of both would allow the DOA to be determined, as shown by Hawkes (1998). The Microflown Ultimate Sound Probe (USP) sensor combines three orthogonal particle velocity sensors and one pressure microphone. This enables wideband detection of both the polar and azimuthal angle of the acoustic DOA with just a single sensor with certain limitations which will be discussed.

Caulk (2009) demonstrated this using a hybrid array of two Microflown sensors and one pressure microphone. He successfully determined the DOA of two sound sources operating at two different frequencies in the anechoic chamber to within about 5^0 accuracy. However, experiments on contacts of interest under field conditions failed due to wind noise issues.

This paper builds upon Caulk's thesis to demonstrate the Microflown USP's ability to track actual planes during takeoff and landing at an airport. Advances include the direct extraction of sensor data into Matlab for processing (without having to use LabView), and a near real-time display of DOA as visual feedback. A foam wind shield is also utilized to overcome the wind noise that prevented the USPs from tracking actual targets previously.

The rest of this paper is organized as follows:

- Chapter II: Theoretical background to vector sensors and their beamforming.
- Chapter III: Hardware and software used, details of apparatus and experimental setup.
- Chapter IV: Signal processing algorithms and considerations.
- Chapter V: Results and analysis.
- Chapter VI: Conclusion and recommendations.

II. THEORY

A. AXES SETUP

Figure 1 shows a close-up of the Microflown USP acoustic vector sensor, together with the axes configuration for this thesis. The three particle velocity sensors (BLUE, GREEN, RED) are orthogonally placed to detect sound from the corresponding Cartesian axes. ϕ and θ are the azimuthal and polar angles respectively.

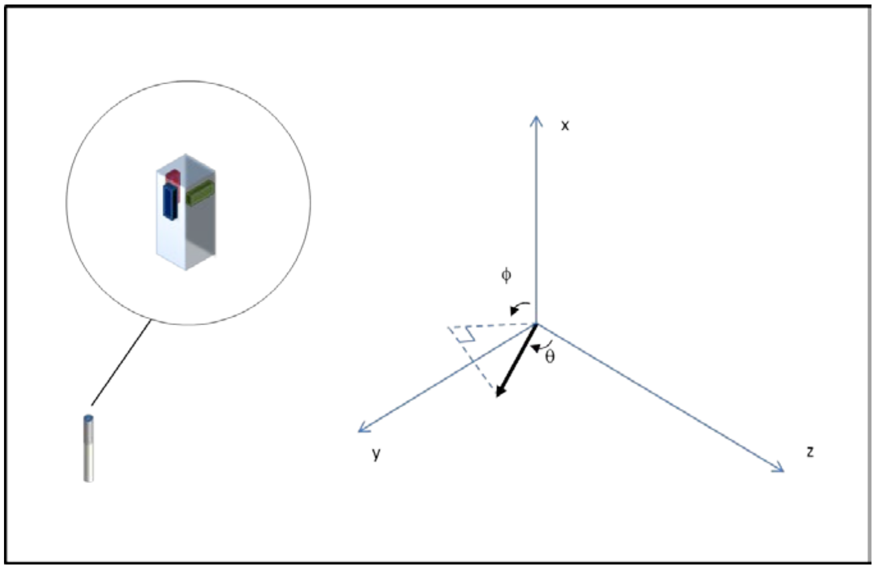

Figure 1. Schematic close-up of Microflown USP and axes setup

B. BEAMFORMING VECTOR SENSORS

Each vector sensor includes four sensors, one for pressure and three for particle velocity:

e_1 is the pressure sensor output

e_2 is the BLUE velocity sensor output (z-axis)

e_3 is the GREEN velocity sensor output (x-axis)

e_4 is the RED velocity sensor output (y-axis)

Although there are always slight deviations from orthogonality amongst the three velocity channels, previous work showed that these deviations were small compared to the uncertainty in the beamformer output. Let (θ_2, ϕ_2) be the direction of the Maximum Response Axis (MRA) for the BLUE vector sensor, (θ_3, ϕ_3) be that for GREEN, and (θ_4, ϕ_4) be that for RED. The unit vectors corresponding to the MRA of each of the sensors would then be given in terms of the direction cosines as:

Equation Section 2

$$\hat{u}_{e2} = \sin\theta_2 \cos\phi_2 \hat{x} + \sin\theta_2 \sin\phi_2 \hat{y} + \cos\theta_2 \hat{z} = \hat{z} \tag{2.1}$$

$$\hat{u}_{e3} = \sin\theta_3 \cos\phi_3 \hat{x} + \sin\theta_3 \sin\phi_3 \hat{y} + \cos\theta_3 \hat{z} = \hat{x} \tag{2.2}$$

$$\hat{u}_{e4} = \sin\theta_4 \cos\phi_4 \hat{x} + \sin\theta_4 \sin\phi_4 \hat{y} + \cos\theta_4 \hat{z} = \hat{y} \tag{2.3}$$

It should be noted that various beamforming methods exist, and this thesis retains the method used by Caulk. Prior to beamforming, the outputs of the velocity sensors must be adjusted to be commensurate with the amplitude and phase of the pressure sensor output. This is done by measuring the transfer functions between the pressure and particle velocity outputs. While this can theoretically be derived via the supplied calibration data of the Microflown USPs, it may fail to capture effects of our mounting apparatus, or other sources of interference or noise from the rest of the electronics. A comprehensive method is to conduct in-situ calibration with the sensor in its experimental setup, which will be discussed in the following chapter.

Once the outputs of the individual sensors have been properly calibrated, they can be put through the beamforming algorithm. For a steer direction of (θ_s, ϕ_s), the unit vector in the steer direction is given in terms of the direction cosines as:

$$\hat{u}_s = \sin\theta_s \cos\phi_s \hat{x} + \sin\theta_s \sin\phi_s \hat{y} + \cos\theta_s \hat{z} \tag{2.4}$$

Each particle velocity output would then be weighted accordingly using this steering vector by taking their dot products:

$$w_n = \hat{u}_s \cdot \hat{u}_{en} \qquad n = 2,3,4 \tag{2.5}$$

The pressure output is omnidirectional, hence:

$$\hat{u}_s \cdot \hat{u}_{e1} = 1 \tag{2.6}$$

Next, the velocity channels need to be expressed in terms of their equivalents to the pressure signal. This can be achieved by applying the appropriate transfer functions, described more in the next section. Eventually when given $\hat{X}c_n(k)$, the discrete Fourier transform of e_n corrected to the equivalent of the pressure sensor, the linear (Bartlett) beamformer output is given by:

$$B(k,\theta_s,\phi_s) = \sum_{n=1}^{4} \hat{X}c_n(k) w_n \tag{2.7}$$

$$B(k,\theta_s,\phi_s) = \left| \sum_{n=1}^{4} \hat{X}c_n(k)(\hat{u}_{en} \cdot \hat{u}_s) \right| \tag{2.8}$$

For a chosen k^{th} frequency bin, the corresponding steer vector u_s that generates the largest $B(k,\theta_s,\phi_s)$ would give the DOA of the incoming signal. Note (2.8) is only valid for a single vector sensor. Multiple sensors would require additional beamforming to factor in the arrival phase differences that would vary according to the sensor separations.

C. MEASUREMENT OF TRANSFER FUNCTIONS

1. Set up

The set up was performed in the anechoic chamber to minimize interference from reflections. The Microflown USP was rigged up in the array[1] holder, and all three sensors were capped with their respective foam windshields, as they would be during data collection.

2. Procedure

The transfer functions for each of the three velocity sensors (BLUE, GREEN, RED) had to be determined separately. Therefore, the sensor had to be rotated each time so that the MRA of the each velocity sensor faced the source speaker. Broadband white noise was generated using a signal generator. Data from all sensors were collected via the National Instruments (NI) cDAQ-9172 and processed in Matlab.

3. Computation

The required transfer function estimates are then generated in the frequency domain by taking the ratio of the cross-spectral density estimate between pressure (channel one) and the particle velocity channel (channel n) and the auto-spectral density estimate of the particle velocity channel:

$$\left\langle \hat{H}_n(k) \right\rangle = \frac{\left\langle \hat{X}_1(k)\hat{X}_n^*(k) \right\rangle}{\left\langle \hat{X}_n(k)\hat{X}_n^*(k) \right\rangle} \qquad n = 2,3,4 \qquad (2.9)$$

where $\hat{H}_n(k)$ is the transfer function of k^{th} frequency bin, $\hat{X}_n(k)$ is respective digital Fourier transform of the time-domain signals.

[1] This paper's work only uses data from a single Microflown USP, as is sufficient to determine DOA as previously mentioned. However for practical reasons as well as to facilitate future work, the hybrid array holder made by Caulk (2009) is retained in the calibration process.

Figure 2 shows the plot of the transfer functions obtained for Sensor 323. The coherence subplot indicates that the usable frequencies begin from approximately 200 Hz onwards.

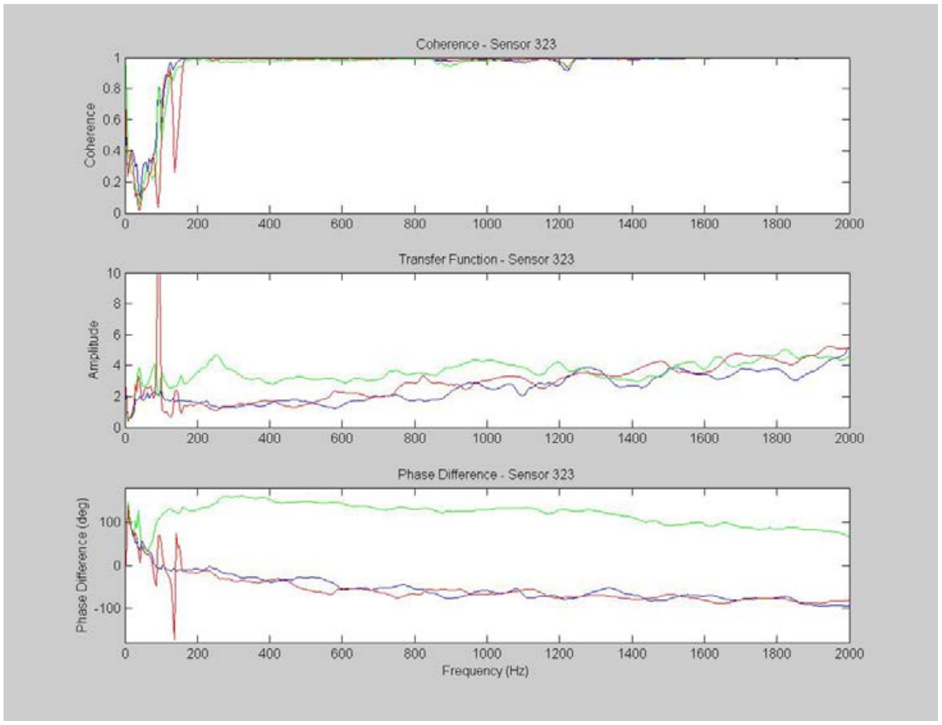

Figure 2. Plot of transfer functions for Sensor 323

THIS PAGE INTENTIONALLY LEFT BLANK

III. EQUIPMENT AND SETUP

A. HARDWARE

Appendix A provides a list of the hardware used. Table 1 lists some important configurations used for this thesis.

Throughout entire study	
Microflown Four Channel Signal Conditioner	- Gain set to 'HIGH' - Correction mode set to 'OFF'
Deriving Transfer Functions	
AUSTIN AU-15G Amplifier	- Overdrive set to 'ON' - Volume set to 'MAX'
HP Signal Generator	- White Noise selected - Vpp set to '5V'
Windshield Testing	
Kestrel 4000 Handheld Anemometer	- Data recording mode at 2-sec interval

Table 1. Hardware configurations

B. SOFTWARE

Previous work done by Caulk (2009) had sensor data recorded via LabView, and subsequently post-processed in MatLab. This precluded real-time feedback for in-situ evaluation and adjustments if necessary. To overcome this, the latest Data Acquisition Toolbox within Matlab (version 2010b and above) was utilized to directly acquire data from the cDAQ.

The detailed component requirements are:

- Matlab version 2010b and later
- Data Acquisition Toolbox
- NI DAQmx 9.4 (http://joule.ni.com/nidu/cds/view/p/id/2604/lang/en)

This method also offered additional benefits such as programmatically setting the sampling frequency, auto-detection and processing of signals above threshold strength, etc. The downside however is that the responsiveness is dependent on the computer power available. This study used a Dell Precision M4500 laptop (Core i7 1.60GHz, 4 Gb RAM, 64-bit Windows 7) and was able to run the code continuously at a refresh rate of 0.5s or less.

C. EXPERIMENTAL SETUP

1. Anechoic Chamber

For preliminary tests of the algorithms in the anechoic chamber, all of the equipment used for deriving the transfer functions were retained, except for the source. A bookshelf speaker was used, driven by a signal generator at monotones of various frequencies. Tests were conducted at varying locations respective to the sensor, as well as at different frequencies within the 2 kHz-Nyquist limit.

A second set of tests also included ‘clapping hands’ and ‘humming’ while walking around the sensor array. Attempts were also made to test the system using mini remote-controlled helicopters, although it was observed that they did not generate enough signal strength.

2. Field Measurement

The primary purpose of this study was to acoustically detect and track aircraft. The chosen location was therefore in proximity (~1 mile to nearest edge of runway) to nearby Monterey Airport and positioned such that arriving and departing aircraft would almost fly overhead for maximum signal strengths. The apparatus was mounted on a stand and placed on a concrete path next to a grass patch, approximately 30 meters from the nearest building (Hermann Hall, Naval Postgraduate School). The foam windshields were used on all sensors to minimize wind noise. Figure 3 shows a bird's eye view of the field.

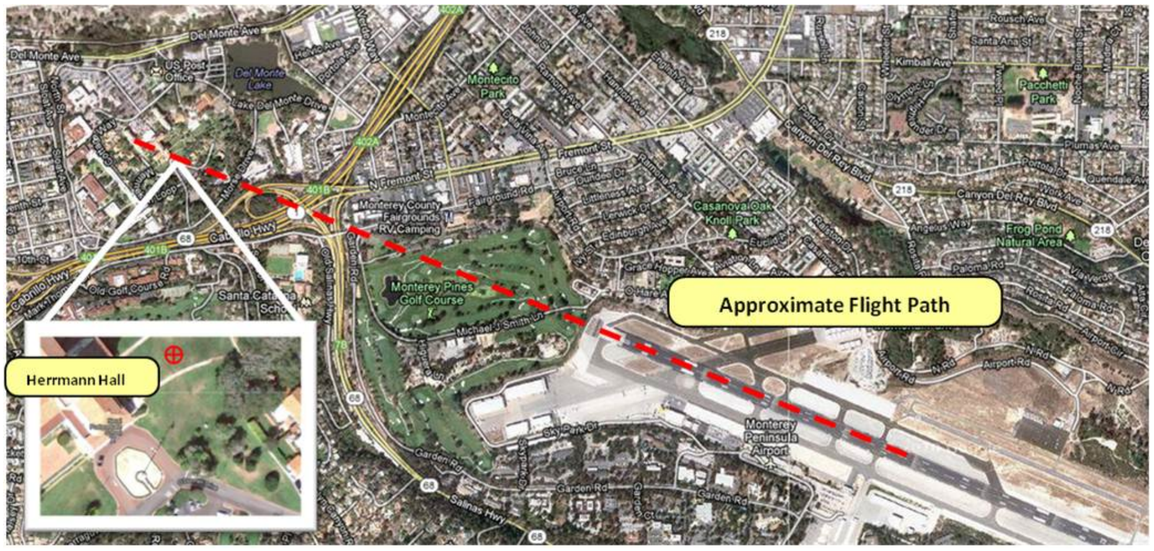

Figure 3. Field location with respect to Monterey Airport's runway

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SIGNAL PROCESSING

This section discusses some of the programming and signal processing considerations. Key concepts and methodologies will be highlighted, together with notes about what worked and what did not.

A. SAMPLING FREQUENCY

The National Instruments Data Acquisition (DAQ) equipment uses a master internal timebase of 13.1072MHz. The sampling rate is controlled by the first installed module with a maximum of 51.2kHz. All other sampling rates must be integer divisions of the maximum according to the formula:

$$f_s = \frac{f_M / 256}{n} \tag{4.1}$$

where f_M is the master clock rate and n is an integer from 1 to 31 (National Instruments, n.d).

For this thesis, a sampling frequency of 4267Hz was chosen (using $n = 12$ and rounded off) to achieve a Nyquist frequency of about 2 kHz. This was adequate for the application to commercial aircraft with noise signals typically ranging from 200 to 500 Hz.

B. MATLAB / DAQ-MX CONFIGURATION

Matlab's Data Acquisition Toolbox works with NI DAQmx to import data from the Microflown USP, via the NI cDAQ in real time. The process is generally straightforward and simple to use. The following discusses some salient points during set up:

1. Listening Mode

Two modes are available; Foreground or Background. The latter was chosen since it allowed simultaneous operations to be carried out while the data was streaming. Foreground mode presumably would be more suited to applications requiring higher degrees of control, such that data processing commences only upon the completion of data capture.

2. Data Stream / Buffer

It was observed that the data stream in Background mode enters a memory buffer, and when deemed 'available' the programmed function will be invoked. It appeared, however, that the 'Data Available' criterion was not consistent either by data size (number of data points) or time. A preliminary conclusion was that this refresh rate was a function of overall computing speed and data capture parameters, possibly including the prescribed sampling rate and even the USB transfer rate. The variance of each sequence was however not more than 10% each time. For example, in one instance the buffer could provide 590 data points, the next 605, then 610.

Nonetheless, the subsequent data preparation and beamforming expects a consistent and pre-determined data length. As a workaround, the data stream was cumulatively assigned to a growing vector, and the latest N length of data would be used each time. The length N would be a function of the desired resolution (longer means better frequency resolution after FFT), noise reduction potential (more data allows noise reduction through averaging) as well as target motion resolution (shorter means a tighter snap-shot in time). This thesis adopted the following:

Length of data processed each time = 2^{10} = 1024

Given a sampling frequency, f_s of 4265 Hz,

Frequency resolution = $\frac{f_s}{N}+1$ **= 4.16 Hz**

Time duration $= \frac{1024}{f_s}$ **= 0.24 s**

3. Real-time processing and feedback

This is a key component of this thesis. To allow at least near real-time feedback, the data stream needs to be processed and its results presented almost immediately. The time allowed for this would naturally vary across applications, although in general the faster change in target position (as observed by sensor), the faster refresh rate necessary.

Depending on the available computer power, no additional setup may be necessary. The Dell M4500 laptop, however, was not able to process every data stream quickly enough. The alternative was therefore to only process data streams at intervals, say every x instances of 'Data Available'. In this case, processing only every 4th data stream resulted in a sustainable refresh rate of less than 0.5 seconds. Each time only the latest N bits of data were used to ensure the most up-to-date data was being processed.

4. Saving Data Streams

Notwithstanding the above, the integrity of the entire input data stream is preserved in the operating environment, and is saved to file for further processing if needed. This allows detailed analysis which may not have been possible during collection.

C. FREQUENCY DOMAIN ANALYSIS

Once the data is properly extracted, most of the processing occurs in the frequency domain. This allows the beamforming algorithms to target specific frequency or frequency bands of interest. The following discusses some techniques employed to enhance results:

1. Bandpass Filtering

The power spectrum of the incoming data can be evaluated in real-time, and a suitable pass band can be determined through a combination of prior programming bias, and/or rules such as maximum energy detection. This thesis employed a pre-determined pass band of 300–600 Hz, which corresponded to most signals emitted by commercial aircraft. Since the data has already been transformed to the frequency domain, this is a simple process of discarding data outside the pass band.

2. Peak Detection

Within the pass band, a peak detection algorithm determines the peak frequencies for further evaluation. The algorithm compares each bin value to that of its neighbors, and classifies those above the preset threshold as peaks. Threshold settings are adjustable for further refinements.

3. Beamforming

This is the main process of taking the dot products between the MRAs of various particle velocity channels and the unit vector in the steered direction. As expected, this part is the single largest computational bottleneck of the entire program. The more frequencies selected for beamforming and the finer the angular resolution, the longer it takes and slower the overall refresh rate.

Following the peak detection routine, various approaches are possible:

a) Single maximum peak (plus buffer above and below)

b) All detected peaks (plus buffer above and below)

c) Any arbitrary pass band

Initial configurations to achieve the less-than-0.5 second refresh rate adopted Option (a). Option (b) proved to be marginally slower, and Option (c) is highly intensive and more suited to be a post-processing approach.

4. Angular Resolution

The code also allows the angular resolution to be adjustable, and the obvious tradeoff would be between that of accuracy and speed. This was set to be 1-degree increments for the field experiment, and 5-degree increment for the tests conducted in the anechoic chamber.

D. VISUAL DISPLAY FOR FEEDBACK

Last but not least, the results of beamforming has to be presented for feedback. Caulk's work included intensive graphics such as 3D surface plots and full-color imagery. In order to achieve near real-time feedback, this had to be replaced by basic 2-D plots (ϕ - θ), as well as a 3-D directional cone display.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS AND ANALYSIS

A. IN ANECHOIC CHAMBER

1. Single Source, Varying Frequency

Driving a single speaker at various frequencies within the Nyquist limit gives clear and distinct results. Utilizing the single peak detection (Option (a)) also allowed for fast refresh rates and almost instantaneous feedback. The system performed equally well for most frequencies, with the lowest workable frequency found to be approximately 300 Hz. This is slightly higher than the 200 Hz suggested by the earlier coherence plots. The following tests will be presented here:

a) 1000 Hz along positive Y-axis (Figure 4)

b) 1000 Hz along positive Z-axis (Figure 5)

c) 300 Hz along positive Y-axis (Figure 6)

The figures show the detected peak in frequency domains, the 2-D plot, as well as the 3-D cone plot intended as visual feedback in real-time. The figure titles indicate the detected frequencies, as well as source positions.

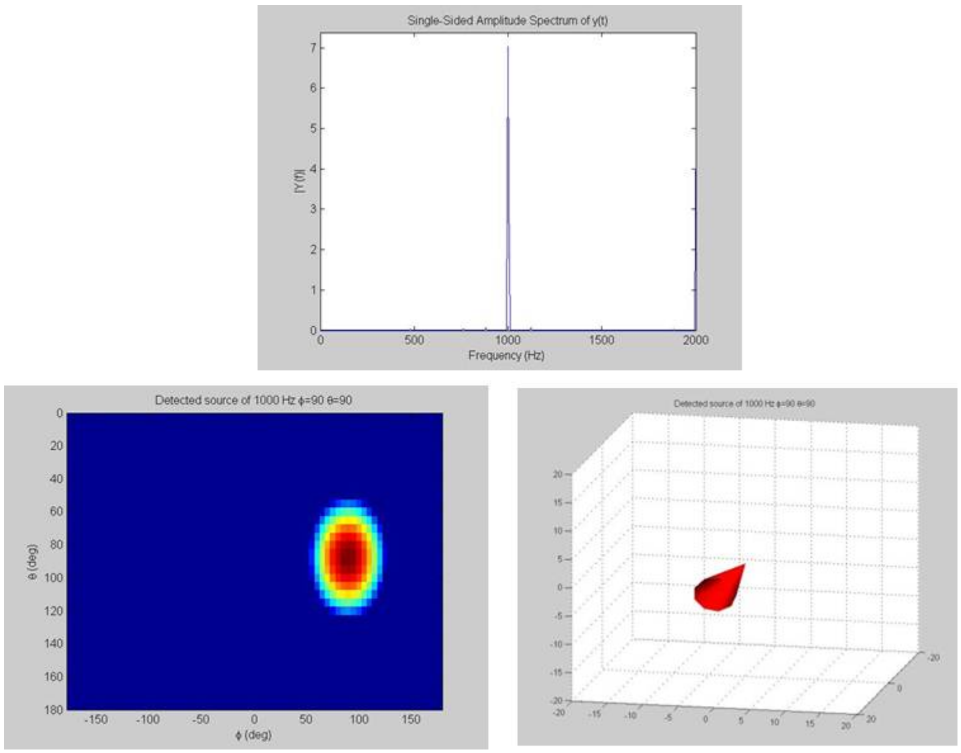

Figure 4. Single Source at 1000 Hz along positive Y-axis

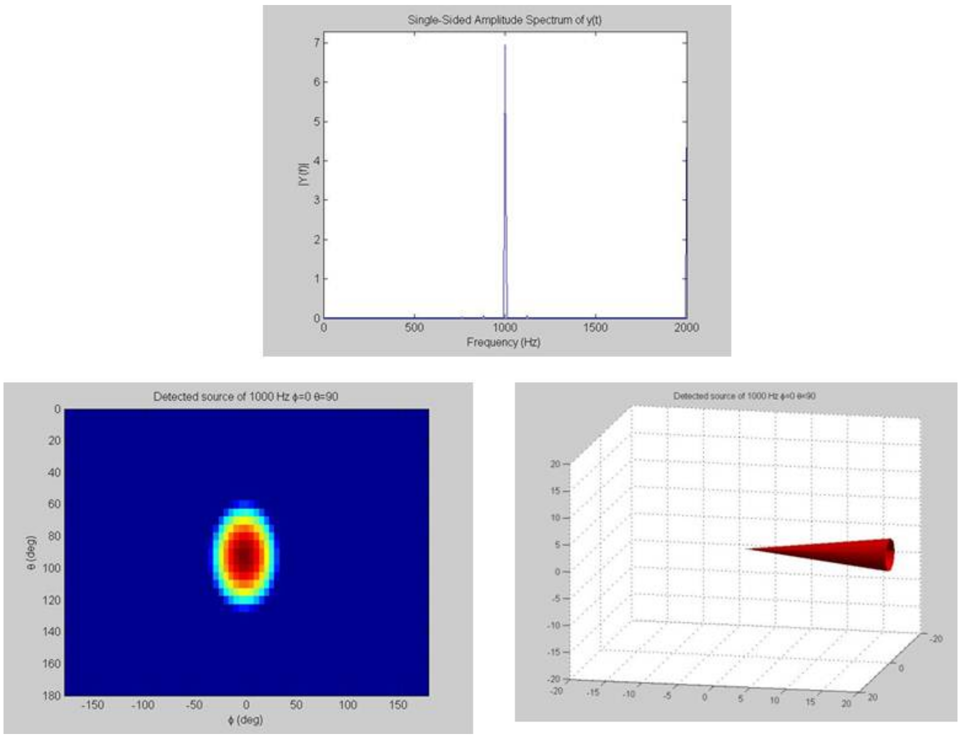

Figure 5. Single Source at 1000 Hz along positive Z-axis

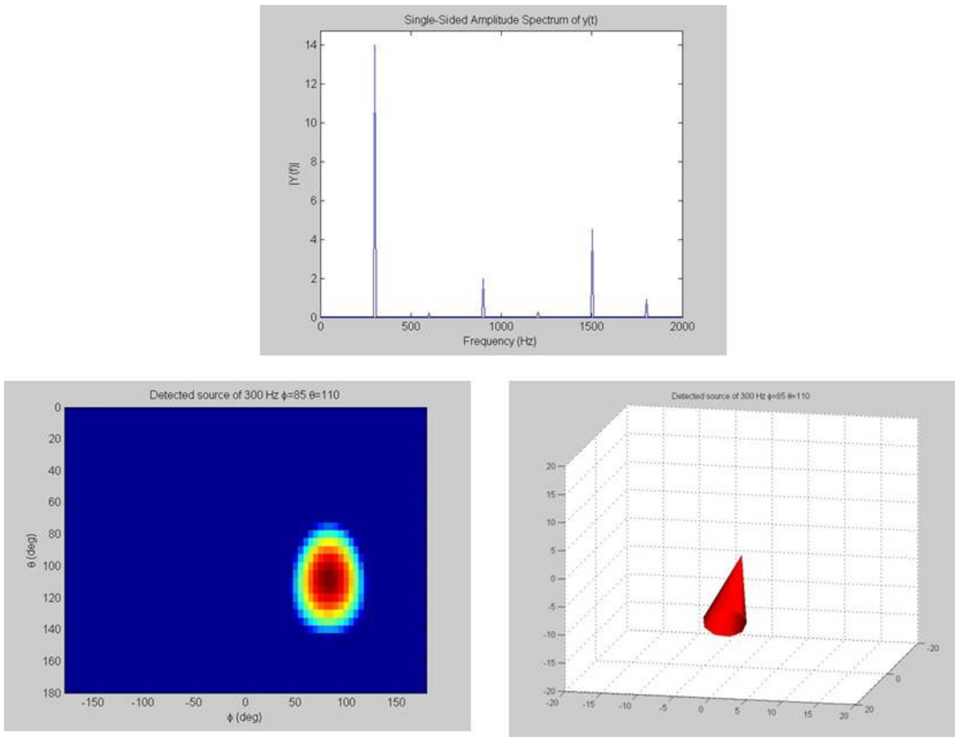

Figure 6. Single Source at 300 Hz along positive Y-axis

While the 1000Hz tone showed excellent tracking along both Y and Z directions, inaccuracies started showing in the tracking of the 300Hz tone. When placed along the positive Y-direction, its detected location was at $\phi = 85°$ and $\theta = 110°$ (5° and 20° error respectively).

2. Two Source, Different Frequency

Two speakers were placed at approximately the same height of and distance away from the sensor array, separated by approximately 50 degrees in azimuth (in the x-y plane). One speaker was driven by a 1000Hz monotone, the other a 300Hz. The system managed to sequentially localize each source, depending on the frequency selection (choice of peak).

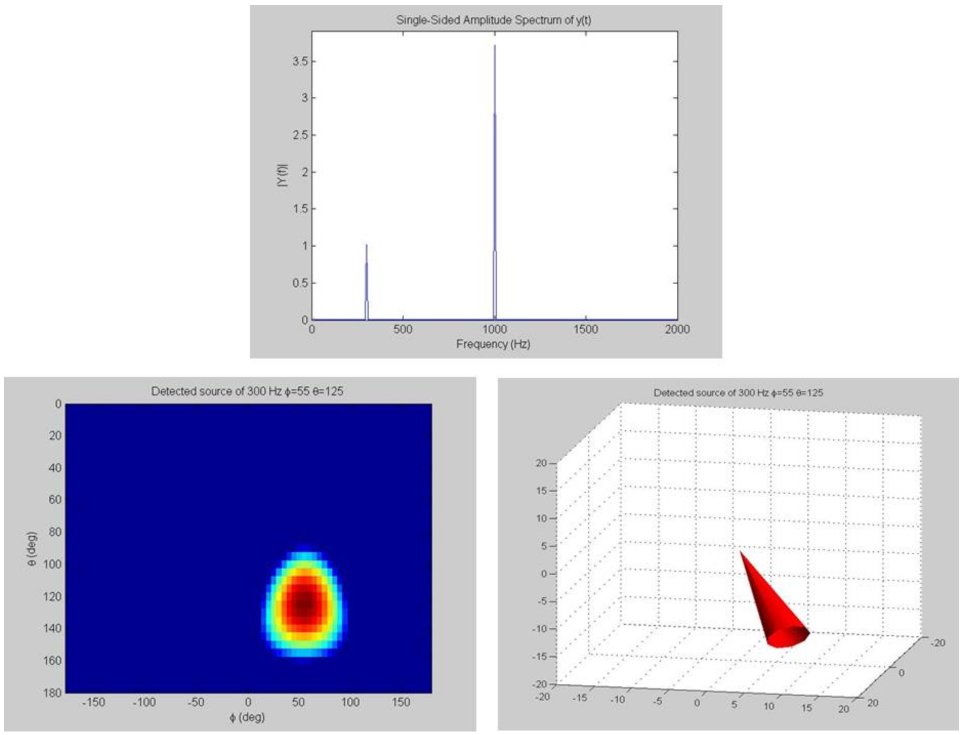

Figure 7. Two sources (300 Hz and 1000 Hz), 300 Hz detected.

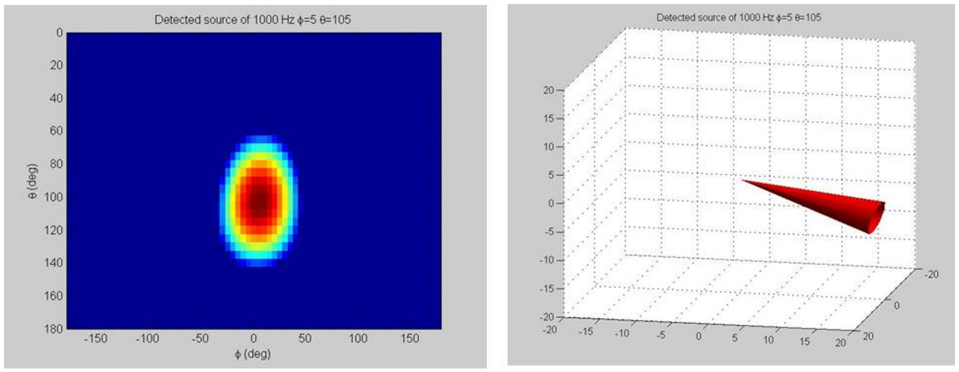

Figure 8. Two sources (300 and 1000 Hz), 1000Hz detected.

Figure 7 and Figure 8 show the correct detection of both monotones emitted simultaneously, albeit with some inaccuracies. The heights of both sources were the same as the previous test (i.e. $\theta = 90°$), but the 300 Hz tone was detected at $\theta = 125°$, while the 1000 Hz tone was detected at $\theta = 105°$.

While it was not done, the system could easily be modified to simultaneously locate and display both source locations transmitting at different frequencies. This would be done via plot-holding in Matlab, and overlaying the two distinct sources as detected by the peak-finding algorithm.

3. Two Source, Same Frequency

As expected and presented by Hochwald and Nehorai, the basic beamforming algorithm employed is unable to distinguish between two sources transmitting at the same frequency or at overlapping frequency bands. A method known as Multiple Source Classification (MUSIC) is widely known to be able to do that. Given *n* vector sensors, the theory hypothesizes the detection of 4*n*-2 uncorrelated sources. A single vector sensor should therefore be capable of distinguishing two sources transmitting at overlapping frequencies.

B. FIELD RESULTS

1. Original Setup

A variety of aircraft were tracked in this session, the two most distinctively different ones are the twinjets and the twin turboprops.

a. Arriving Twinjet

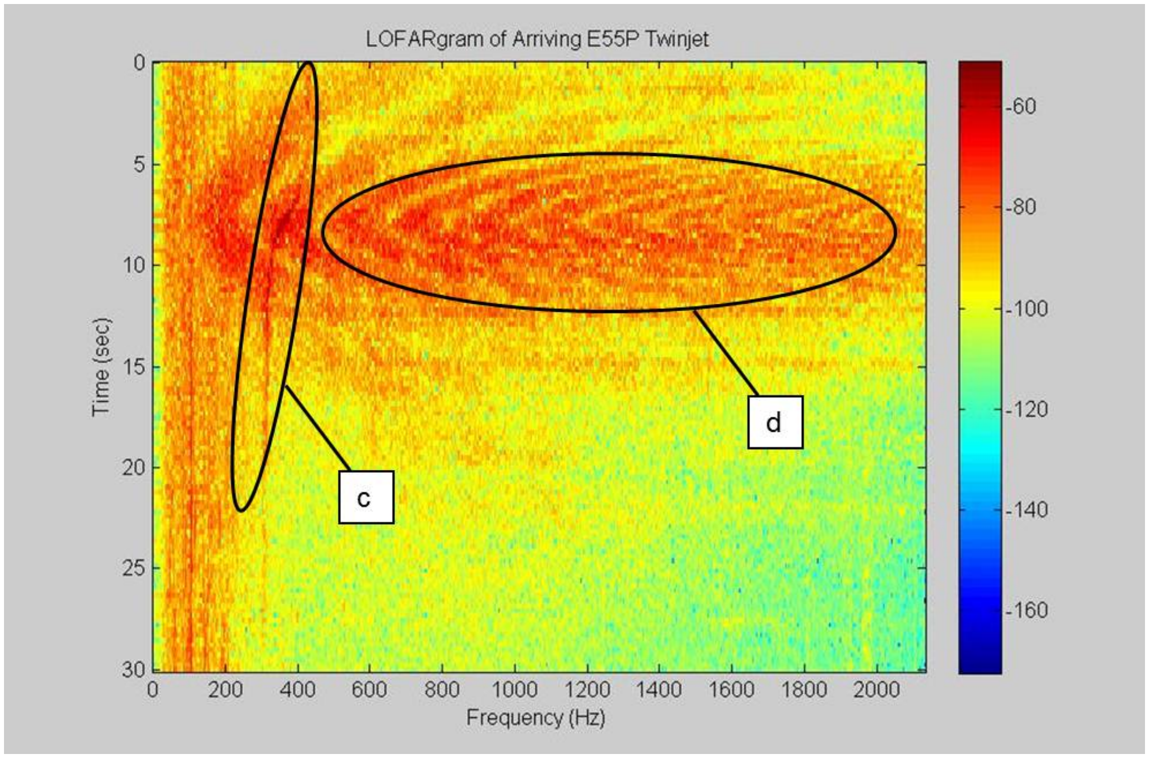

Figure 9. LOFARgram of arriving twinjet

Figure 9 shows the LOFARgram of an arrival twinjet. A few things can be seen immediately from the plot:

a) Target SNR is high and clearly detectable above ambient noise
b) Most ambient noise occur at <200 Hz
c) Clear Doppler shift
d) Distinct Lloyd's Mirror pattern due to interference from reflections

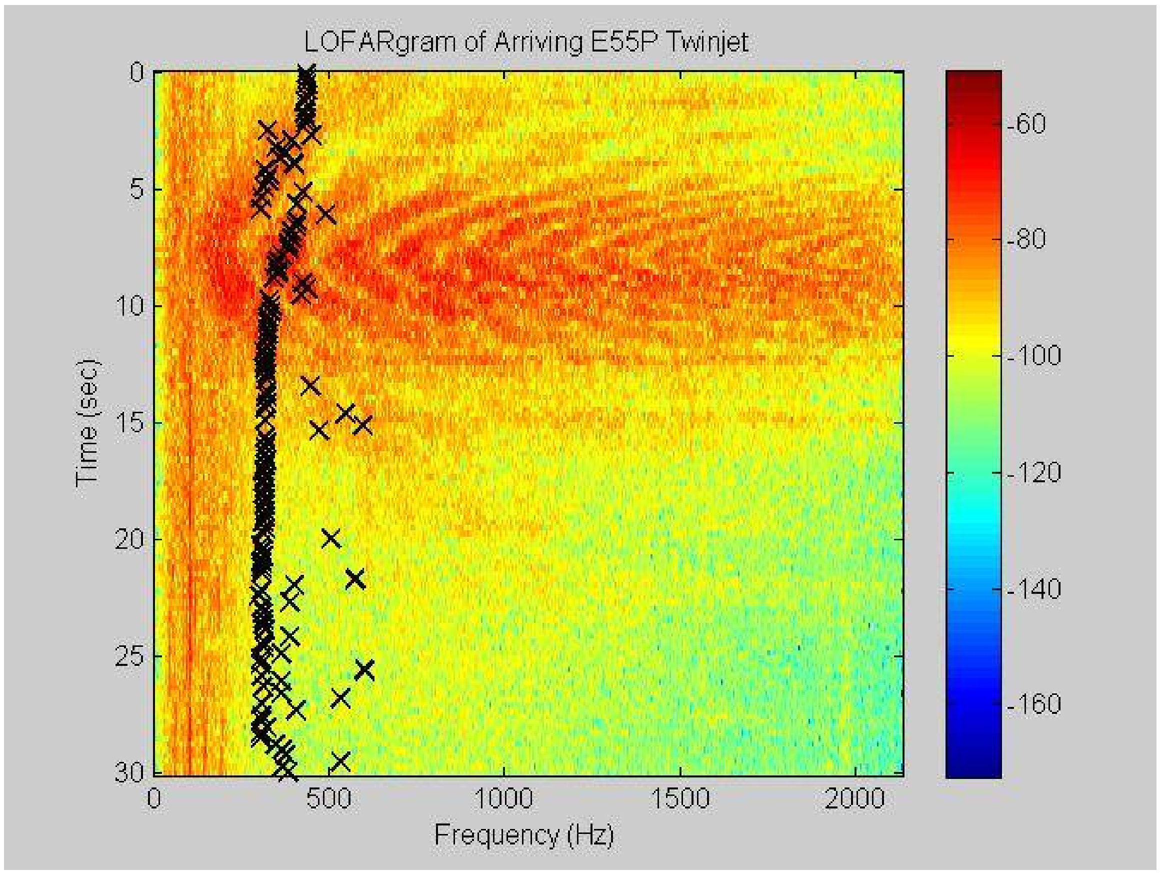

Figure 10. LOFARgram of arriving jet with detected peak frequencies

Figure 10 shows the same LOFARgram, as well as the detected peak frequencies by the peak-finding algorithm, which would be subsequently used for beamforming. Within the specified range of 300–600 Hz, the peak-finding algorithm worked well to identify the main signal frequency most of the time. The detected peaks also helped to clearly identify the Doppler shift.

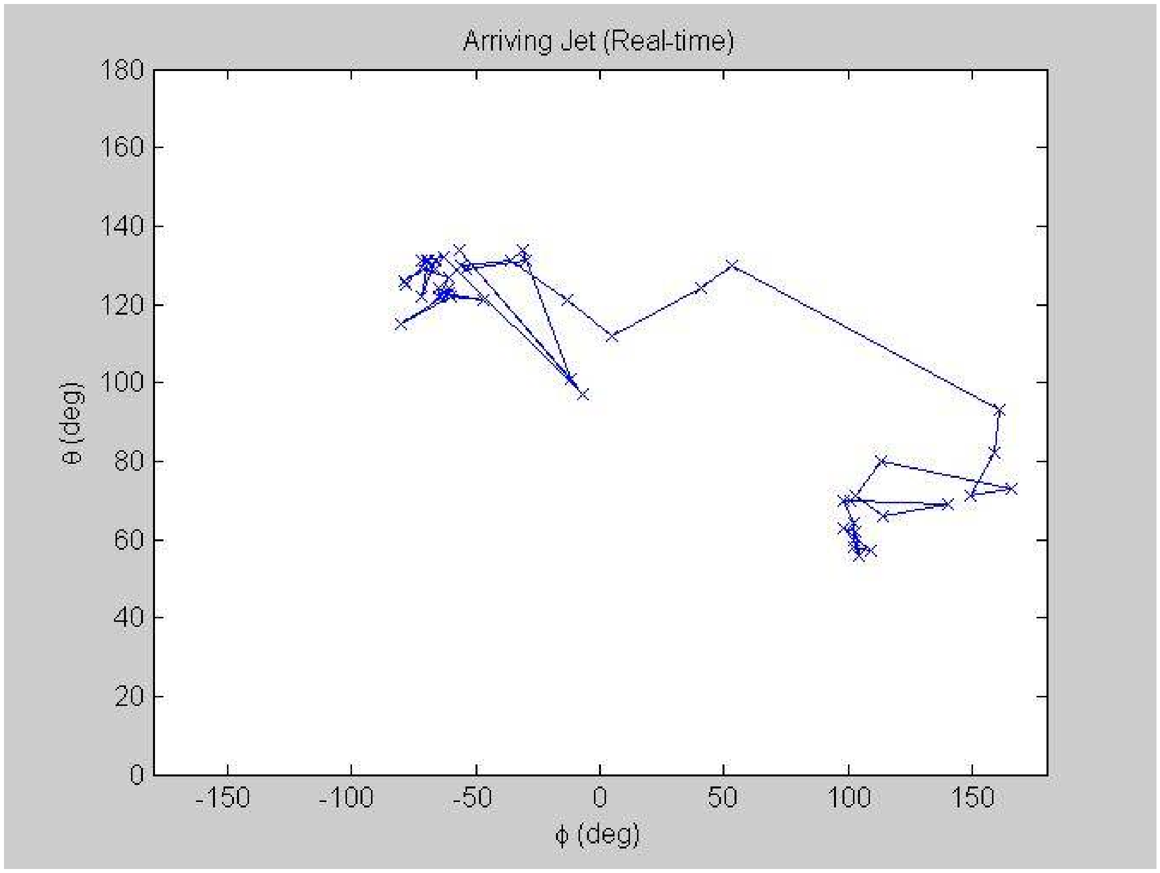

Figure 11. Real-time ϕ - θ plot for arriving jet

The real-time tracking, presented as a 2-D ϕ - θ plot is shown in Figure 11 and was clearly not an accurate representation of the actual flight path. Since the plane was landing, it can be assumed that it was on a straight course. Even with a changing altitude, the actual ϕ - θ plot should be relatively smooth, with small incremental changes in the polar and azimuthal angle.

Since the full data was recorded, post-analysis was possible. The range of beamforming options as earlier presented was tested, and the best result was produced by beamforming across a wider range of frequencies. The following series of plots (Figure 12) illustrate the incremental improvements.

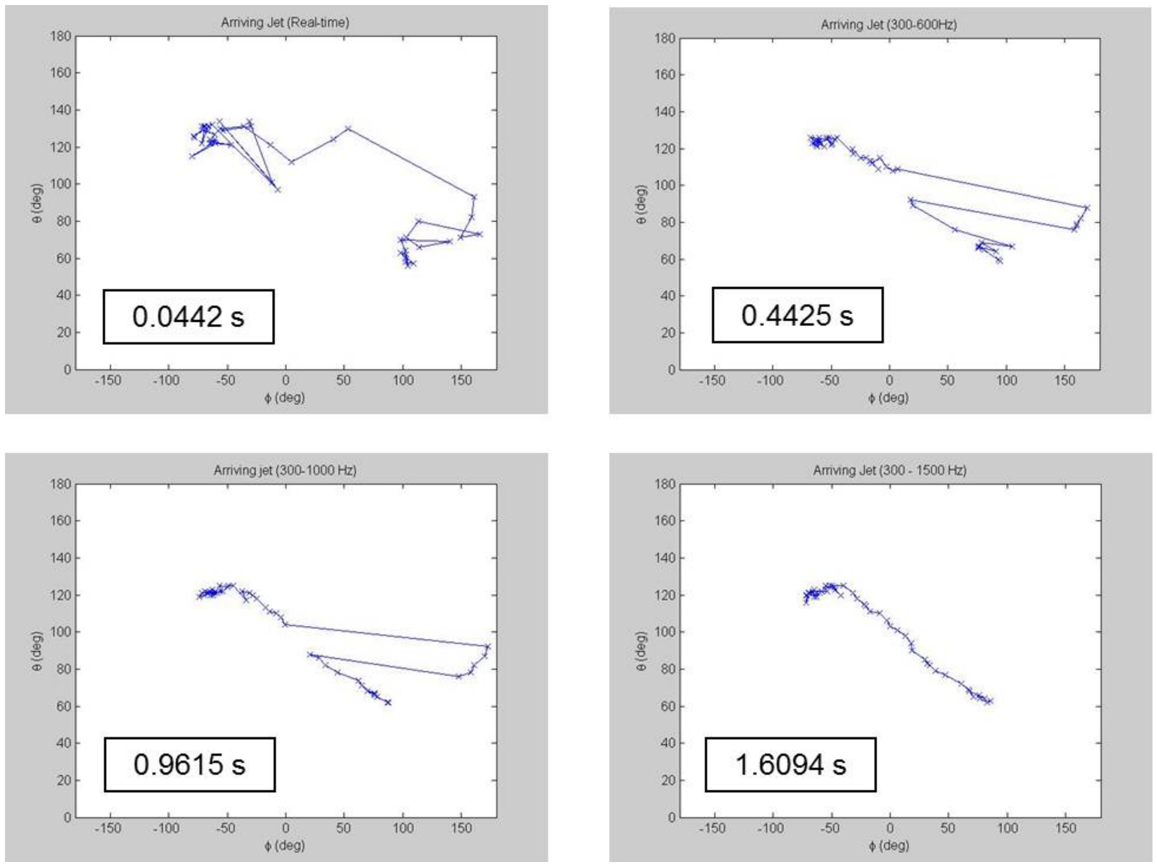

Figure 12. Significant improvements in tracking by beamforming across wider range of frequencies—Arriving Jet

Average beamforming times **per data point/cycle** are appended accordingly. Beamforming across more frequencies essentially perform averaging, and reduces the effect of noise and other interferences. In particular, a vector sensor is most sensitive to reflections that alter the amplitude and phase information of the various channels, and averaging may have mitigated that.

Instead of a selected few frequency bins, the beamformer now computes many more and then sums them cumulatively, before presenting the global maxima for ϕ and θ respectively. This increased computational load would have to be balanced with the desire for system responsiveness.

B. DEPARTING TURBOPROP

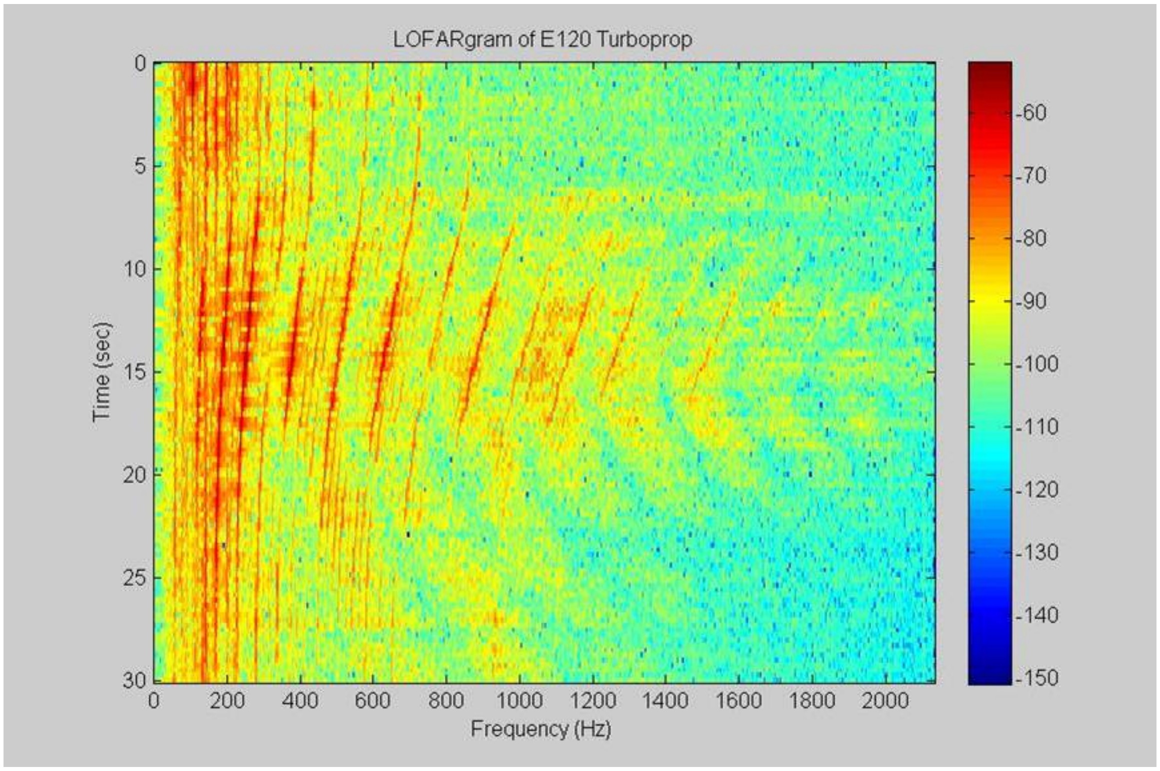

Figure 13. LOFARgram of departing turboprop

Figure 13 shows the LOFARgram of a departing turboprop. As compared to the jet, the broadband signal strength is weaker but still clearly above ambient noise levels. There is still a detectable, albeit smaller Doppler shift due to the apparently lower landing speed. A Lloyd's Mirror pattern can still be seen, but is less obvious due to the smaller signal amplitudes. This is consistent with the in-situ observation that the jets sound louder than the turboprops.

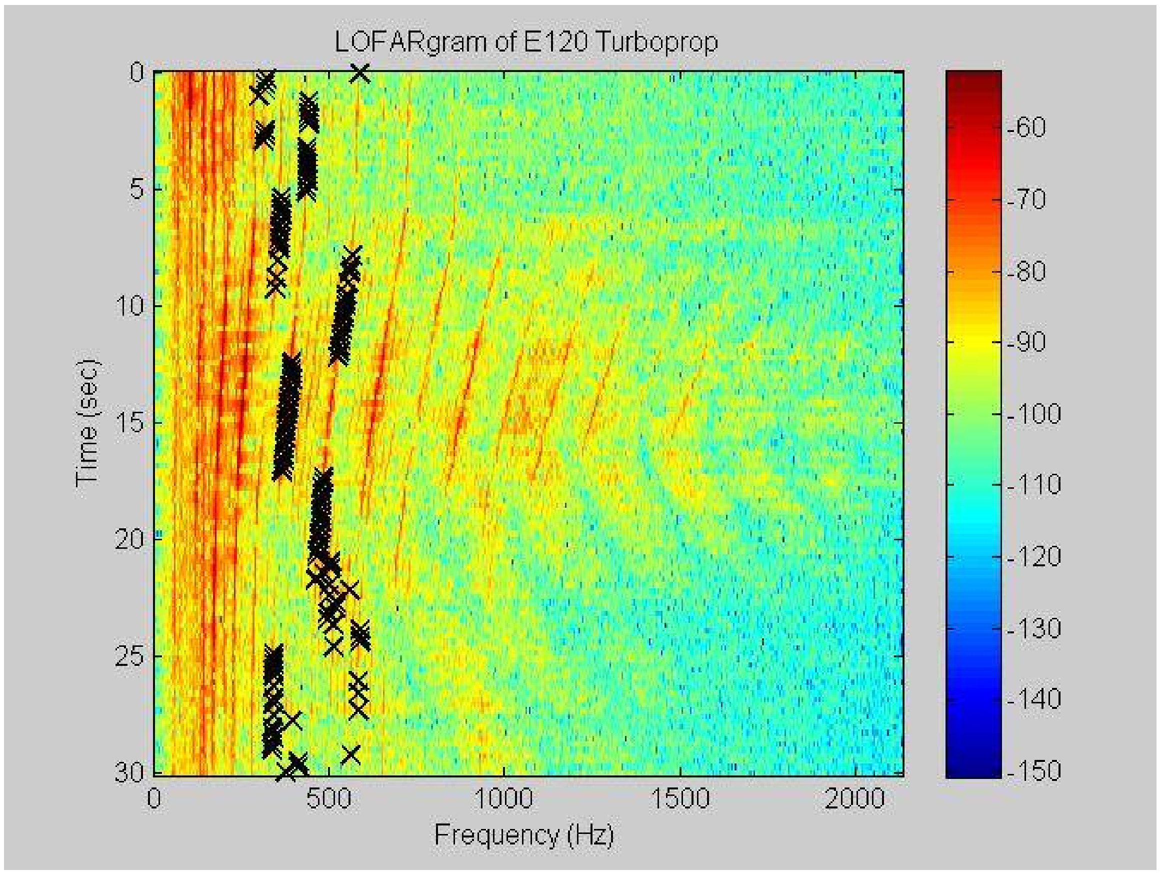

Figure 14. LOFARgram of departing turboprop with detected peak frequencies

Figure 14 shows that the peak-finding algorithm is now struggling to ‘lock’ onto a consistent source. Within the same specified range of 300–600 Hz, the presence of similar-amplitude harmonics appear to be challenging for a single source to be tracked closely. It can also be seen that some high-energy peaks are occurring at around 200 Hz. While expanding the peak-finding algorithm would produce a better ‘lock’ on these emissions, earlier tests had shown the beamformer to only perform decently at frequencies above 300 Hz.

That said, all the detected peaks are without doubt transmissions originating from the target, and not elsewhere. Theoretically, a vector sensor should therefore still be able to resolve the target position from these frequency peaks.

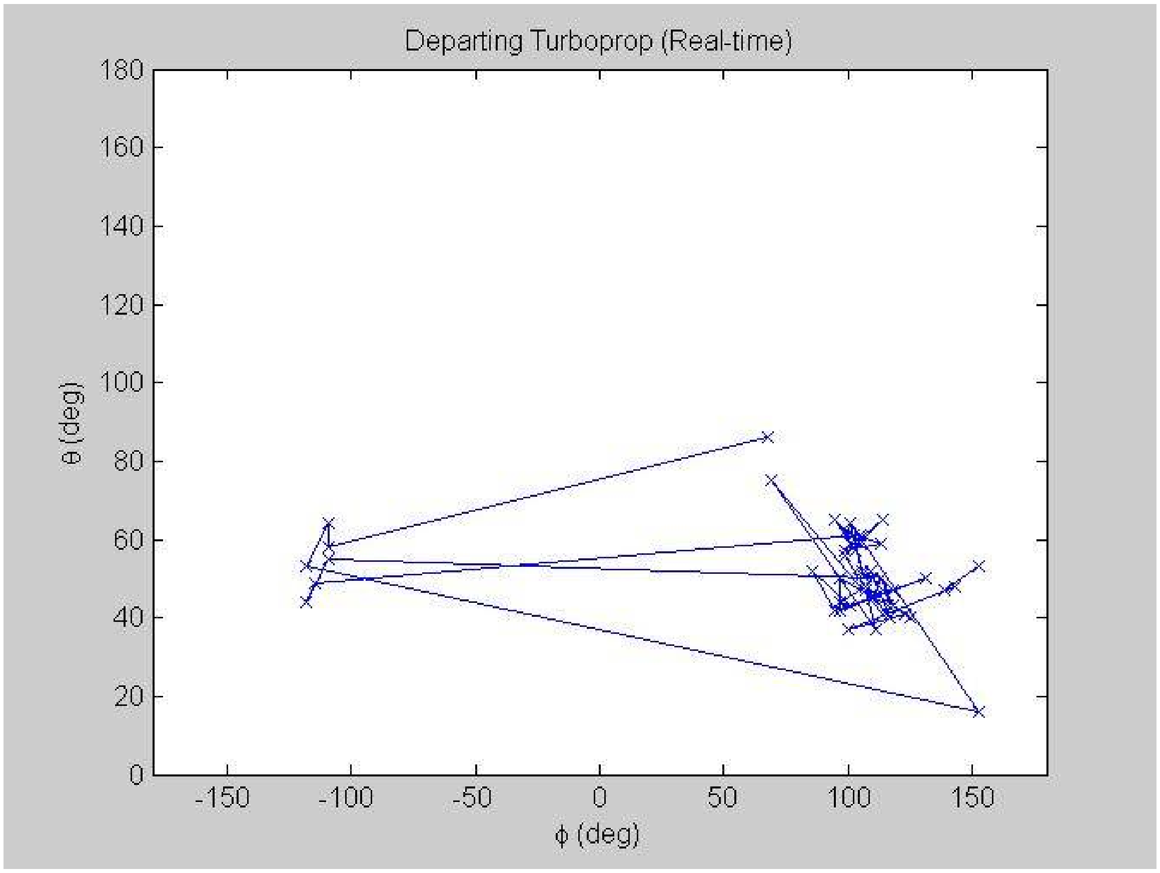

Figure 15. Real-time ϕ - θ plot for arriving turboprop

As seen in Figure 15, the tracking of the turboprop in real-time was also not accurate. Various other beamforming options, including the proven method of summing across more frequencies in the arriving jet did not produce similar improvements. This is shown in the following series of plots (Figure 16).

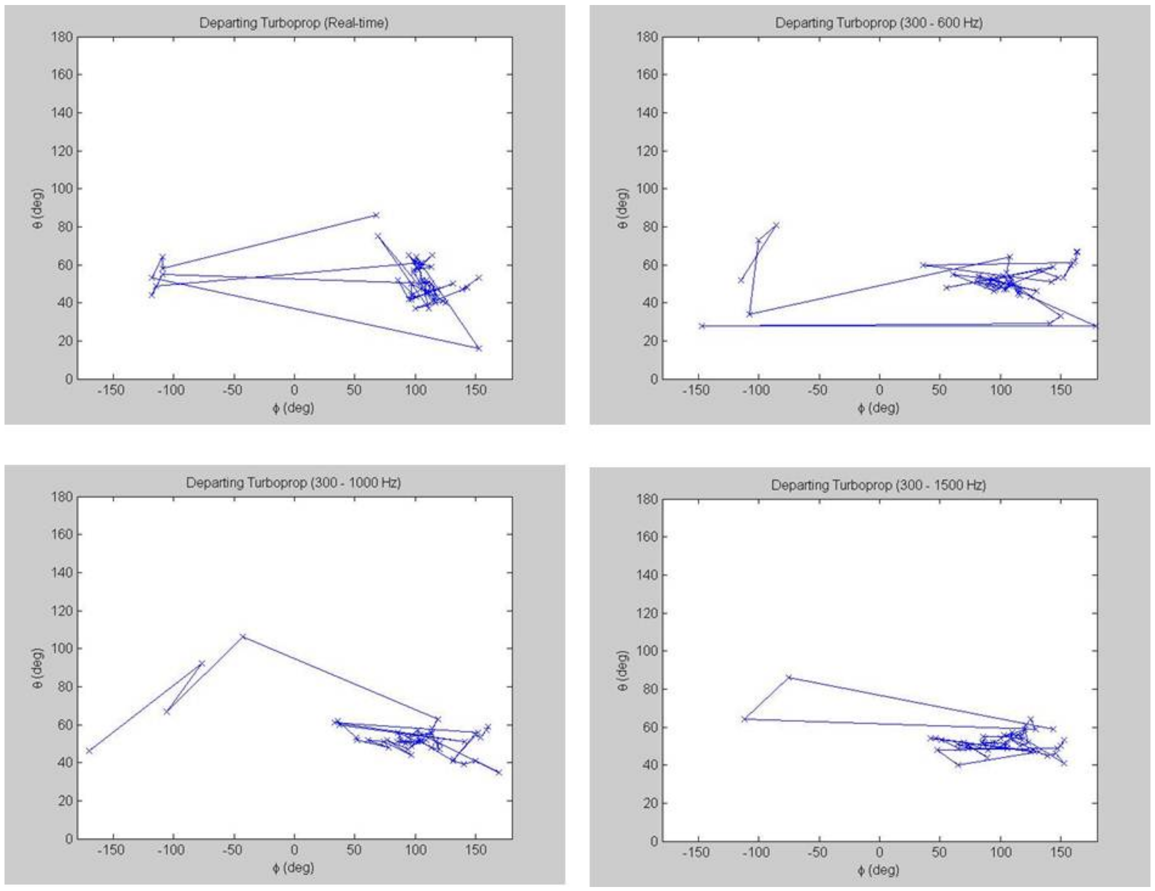

Figure 16. Negligible improvements by beamforming across wider range of frequencies—Arriving turboprop

A few reasons could explain this, the most likely of which is that the highest energy emissions occurred below the minimum working frequency of the Microflown USP. The higher harmonics may not be adequately strong to provide the correct beamforming, or were severely affected by reflections.

2. Revised Setup—with Foam Box

Following the results from the first field setup, an attempt was made to minimize bottom reflections. A cardboard box, lined with sound-absorbent foam was used to shield the bottom half of the sensor. To simplify matters, the sensor array was also replaced by a single Microflown USP on an equipment stand. Figure 17 shows a picture of the revised setup.

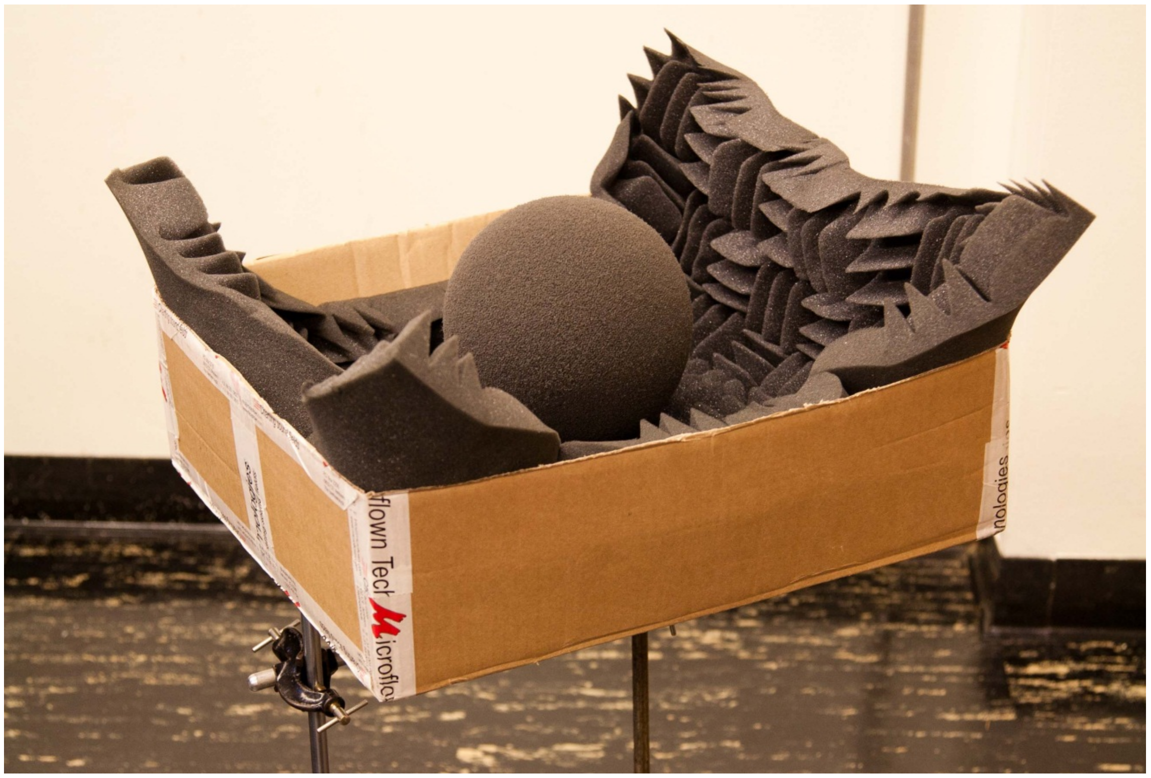

Figure 17. Revised setup of single Microflown USP and anti-reflection foam box still need picture

Since the transfer functions had been derived for the sensor array, the new setup technically had to be re-calibrated by performing the same procedure in the anechoic chamber. A quick check in the chamber however verified that the old transfer functions were still valid, and the new setup could still track sources accurately. The re-calibration was therefore not performed in the interest of time.

a. Arriving Twinjet

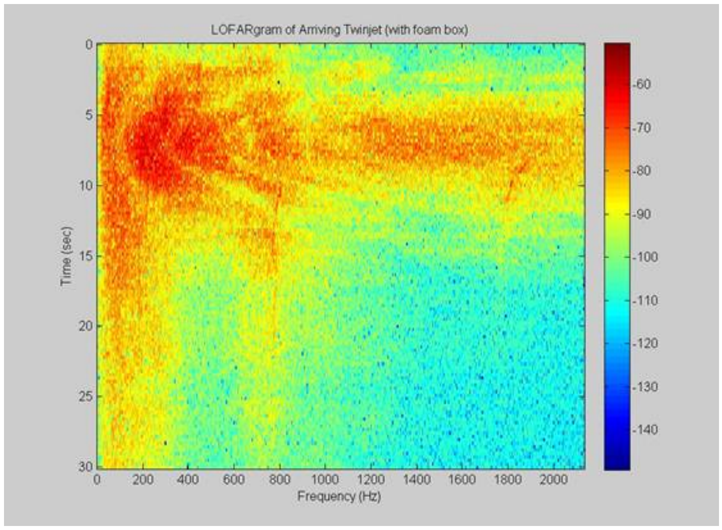

Figure 18. LOFARgram of arriving twinjet (new setup with foam box)

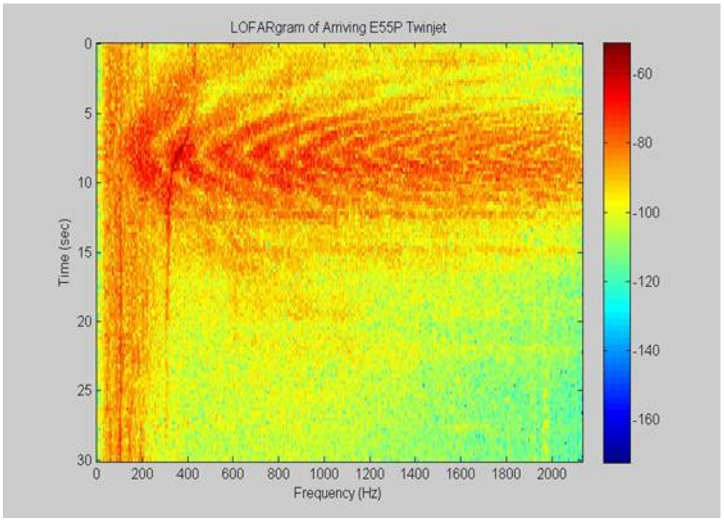

Figure 19. LOFARgram of arriving twinjet (original setup)

Figure 18 shows a markedly different signal profile of a twinjet, with a foam box now under the sensor. Compared to the jet profile using the original setup, shown in Figure 19, the Lloyd's Mirror pattern has been reduced substantially, though still noticeable. Other aspects such as the Doppler shift and overall signal amplitude also seem to have been attenuated to some degree.

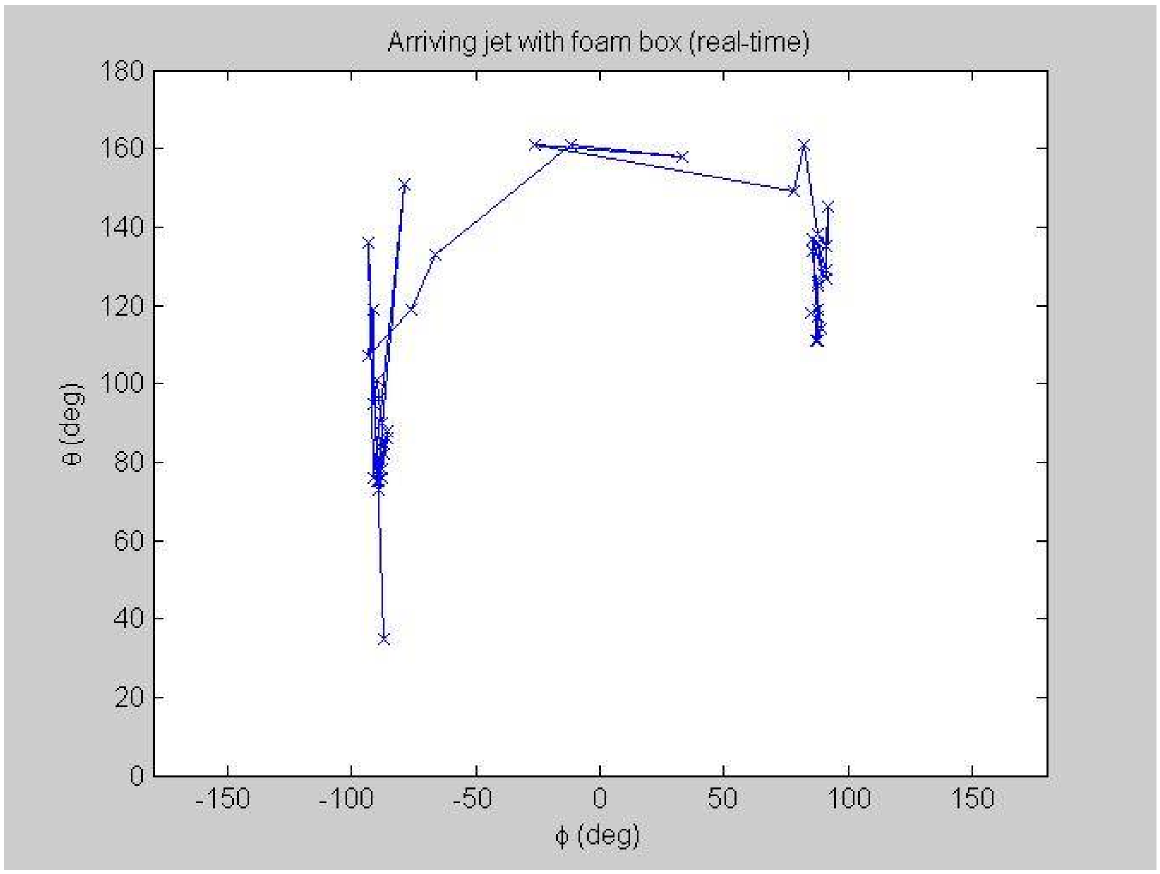

Figure 20. Real-time ϕ - θ plot for arriving twinjet (new setup with foam box)

Figure 20 shows that despite the modifications, the real-time tracking is still not accurate. Post-analysis however show greater incremental improvements when summing across more frequencies.

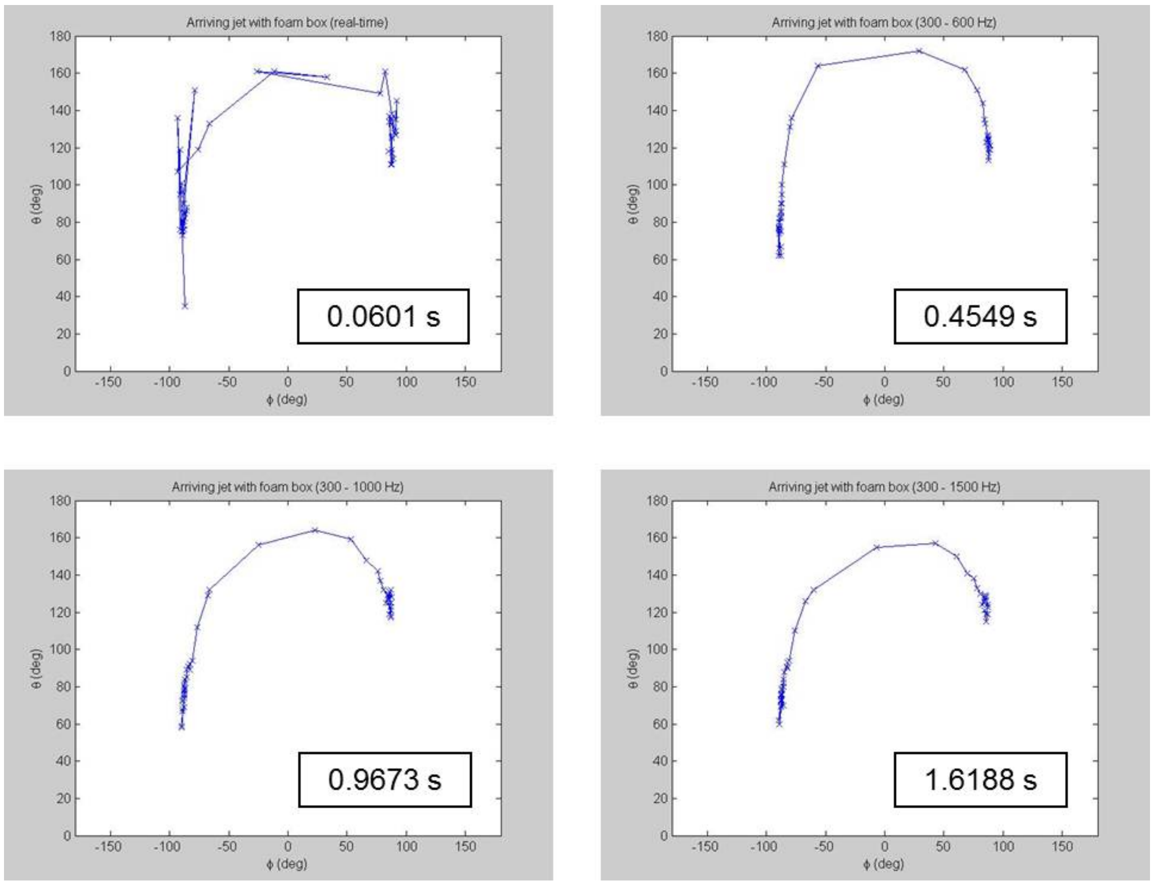

Figure 21. Faster improvements by beamforming across wider range of frequencies—Arriving Jet (with foam box)

Figure 21 shows that the tracking improved substantially faster with beamforming across more frequencies, as compared to the previous setup. The tracking however, while relatively linear and smooth, did not depict the actual flight path. With θ reaching 160° at the closet point of approach (ϕ = 0°), this tracking meant that the flight path was only at a 20-degree elevation from the sensor, when it clearly flew almost overhead. Closer analysis revealed that the the vertical particle velocity sensor (GREEN, x-axis) was recording a less-than-expected signal amplitude. This produced the erroneous tracking of a low-flying target.

It is unclear what may have caused this, but suspicions point to the newly added foam-lined box underneath the sensor. The presence of a faint

Lloyd’s Mirror pattern shows the box is not completely opaque to the signals, it may have in other ways interfered and contributed to the specific attenuation to the GREEN channel. Preliminary research showed that the wedge foam used was not particularly effective in absorbing the low frequencies of 250-500 Hz. If the GREEN signals were artificially boosted by a applying a scalar multiple, the resultant tracking approaches the true flightpath, as shown in Figure 22.

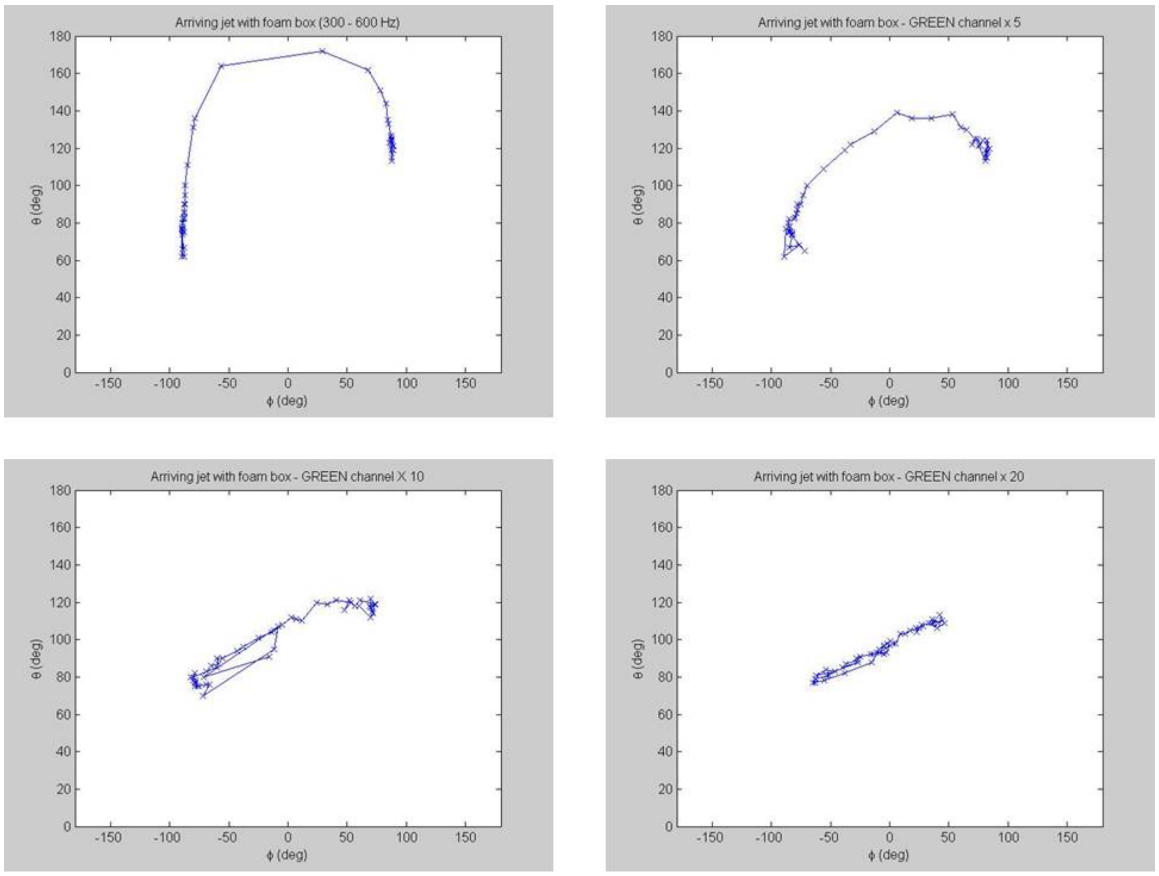

Figure 22. Improved tracking by boosting GREEN signal by 5, 10 and 20 times (beamformed300–600 Hz)

b.　Arriving Turboprop

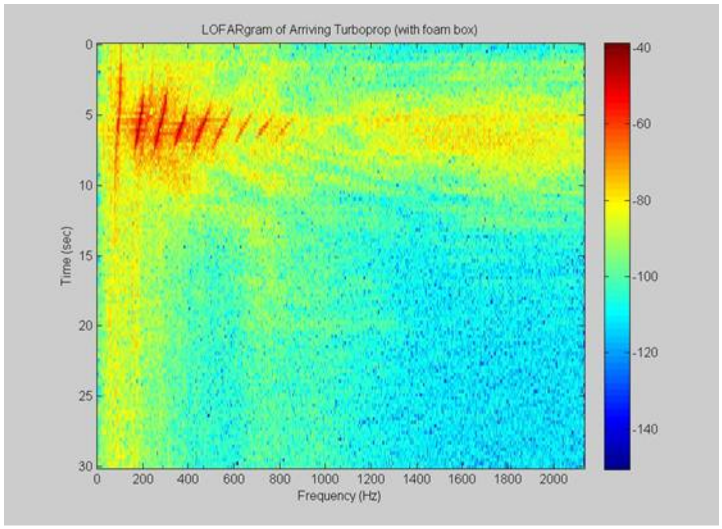

Figure 23.　LOFARgram of arriving Turboprop (new setup with foam box)

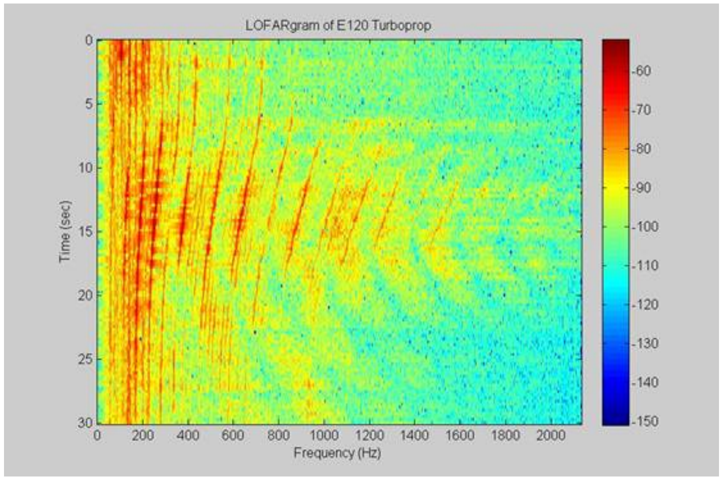

Figure 24.　LOFARgram of departing Turboprop (original setup)

Figure 23 shows the signal profile of an arriving turboprop with the foam box. Compared to that using the original setup in Figure 24, the Lloyd's mirror pattern is noticeably suppressed, especially towards the higher frequencies. Signal strength before and after CPA also appeared to have been reduced to some degree, resulting in shorter frequency 'streaks'.

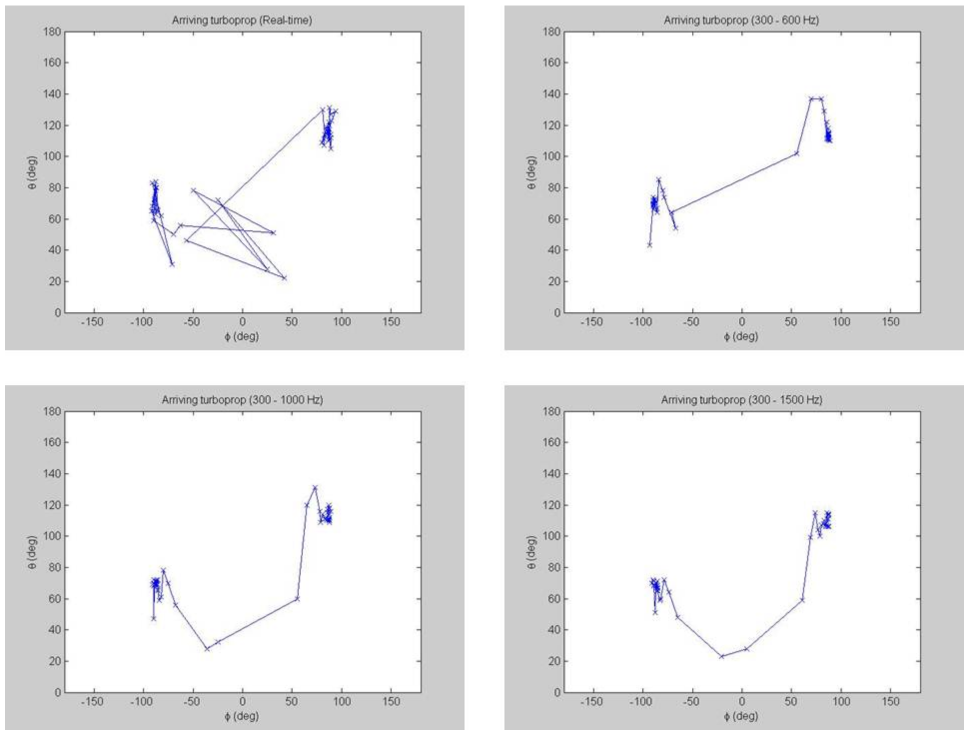

Figure 25. Real-time ϕ - θ plot for arriving turboprop (new setup with foam box)

Figure 25 shows that the real-time tracking again is not accurate. Beamforming across more frequencies also did not give much improvement. Compared to those using the original setup in Figure 16, the tracks, however, were smoother and bore greater resemblance to a realistic flight path.

The 300–1500 Hz plot in particular was interesting. This implied that the plane flew past on the other side of the sensor, away from the buildings ($\theta < 90°$), and at an elevation of 20°. This was previously unseen, and could be due to the reflections off the wall of the cardboard box. Furthermore, the modifications would still not address the fact that some high-energy emissions from the turboprops are occurring below 300 Hz. These would continue to present problems for the Microflown USP since we were unable to produce loud enough low frequency noise to establish good transfer functions that far down in frequency.

3. Analysis of Lloyd's Mirror Pattern

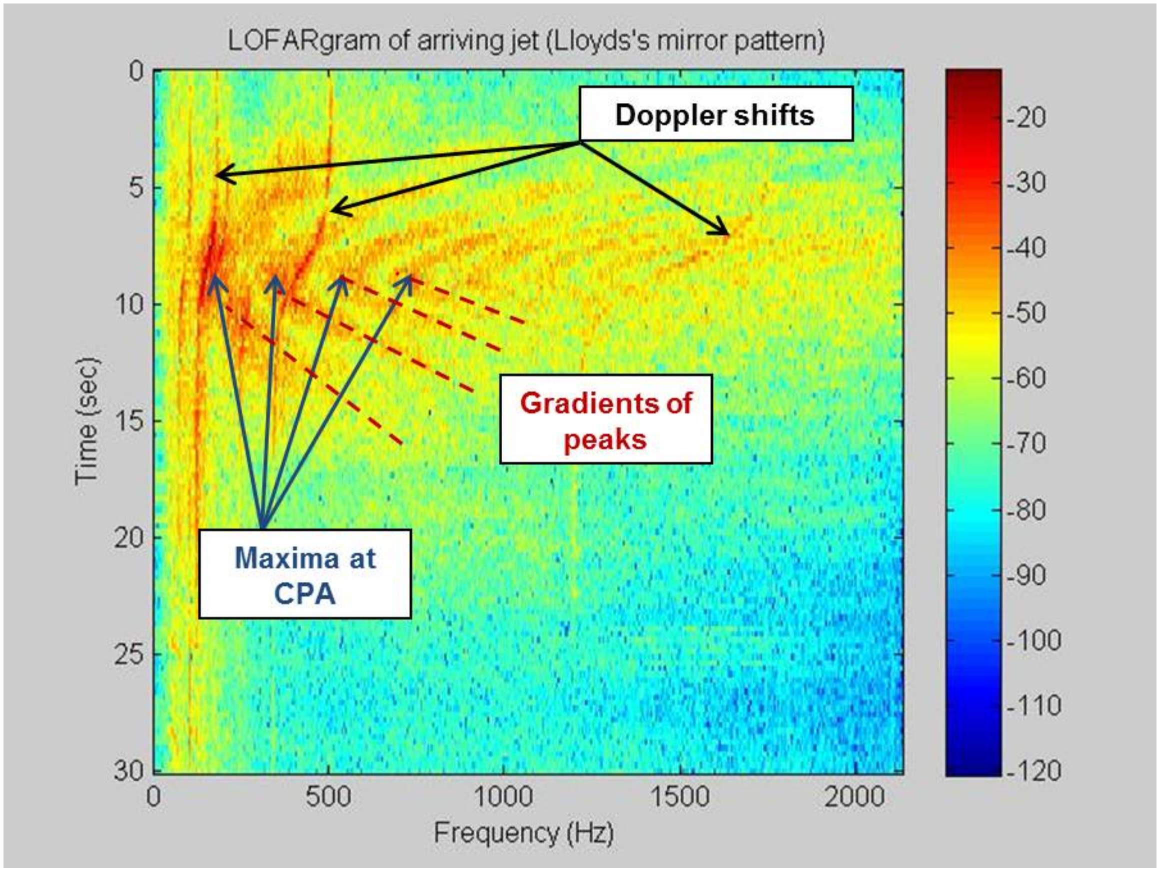

Figure 26. LOFARgram of a arriving jet with distinct Lloyd's mirror pattern

Despite reflections being undesirable for vector sensors in general, additional useful information can be derived from careful analysis. This section explains. The salient features, as indicated on Figure 26 are:

- Doppler shift 'streaks'
- Locations of maxima at CPA
- Gradients of frequency peaks

a. *Target Velocity*

This is the direct application of the Doppler shift equation valid where the contact speed is small compared to the speed of sound:

Equation Section 5 $v = c\dfrac{f_u - f_l}{f_u + f_l}$ (5.1)

where v is the velocity of target, c is the speed sound in air (343 m/s) and f_u, f_l are the upper and lower frequencies of the Doppler streak.

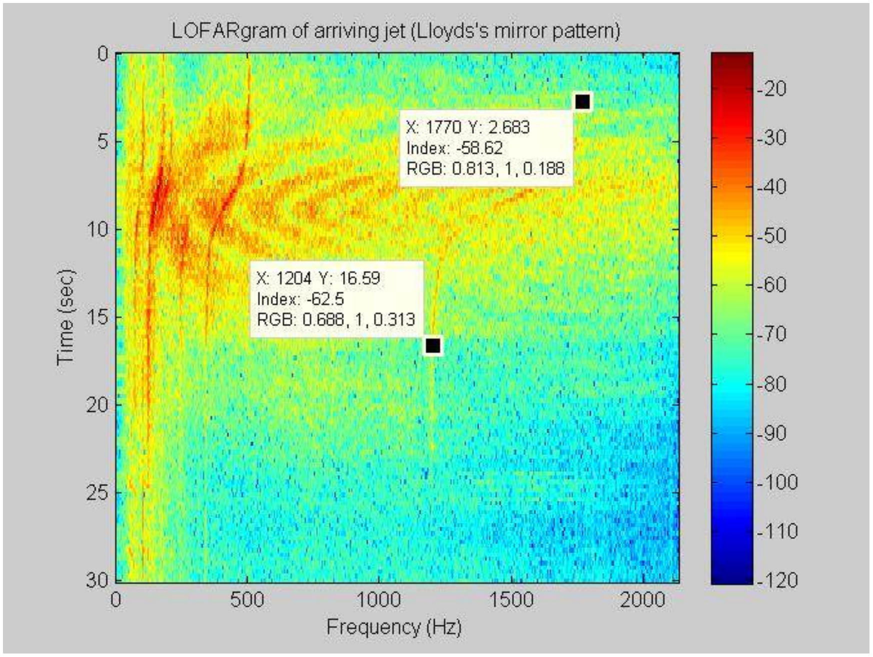

Figure 27. Deriving target velocity from Doppler shift

Choosing the highest frequency visible Doppler streak in Figure 27 for the lowest minimum errors, f_u = 1770 Hz, and f_u = 1204 Hz.

$$v = 343\left(\frac{1770 - 1204}{1770 + 1204}\right) \cong 65\ ms^{-1}(126\ knots)$$

b. Target Altitude

Assume the sensor has a height d, and the aircraft is flying in the y-direction at speed v and altitude h. The horizontal range along the x-axis at CPA is R_0 and occurs at time $t = 0$, as illustrated in Figure 28.

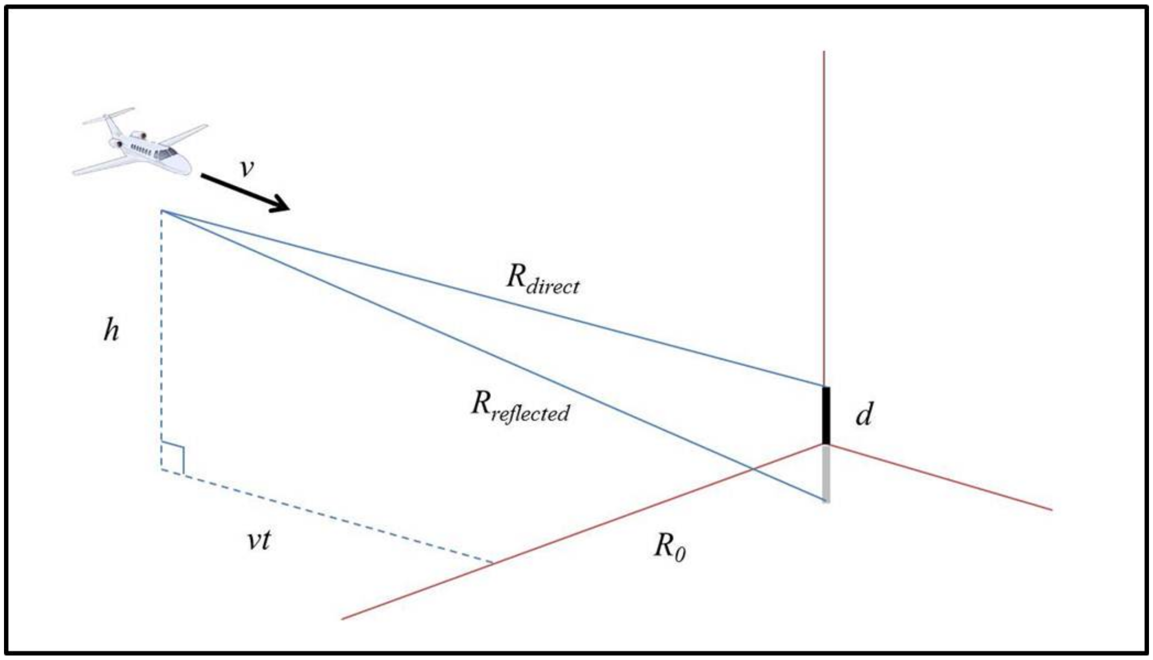

Figure 28. Schematic of flypast

The exact solution for the path length difference between the direct and reflected paths is:

$$\Delta r = R_{reflected} - R_{direct} = \sqrt{\left(h+d\right)^2 + R_0^2 + v^2t^2} - \sqrt{\left(h-d\right)^2 + R_0^2 + v^2t^2} \qquad (5.2)$$

As $t \to \infty$, v^2t^2 becomes large compared to $(h+d)^2 + R_0^2$ and the path length difference can be approximated as:

$$\Delta r = \sqrt{\frac{(h+d)^2 + R_0^2}{v^2t^2} + 1} - \sqrt{\frac{(h-d)^2 + R_0^2}{v^2t^2} + 1}$$

$$\Delta r \cong vt\left(1 + \frac{1}{2}\frac{(h+d)^2 + R_0^2}{v^2t^2}\right) - vt\left(1 + \frac{1}{2}\frac{(h-d)^2 + R_0^2}{v^2t^2}\right) \tag{5.3}$$

Expanding the terms and simplifying results in:

$$\Delta r \cong \frac{2hd}{vt} \tag{5.4}$$

Assuming a rigid boundary condition for the reflection, when the path length equals an integral number of wavelengths, you get maxima in the LOFARgram. This condition is given by:

$$\Delta r \cong \frac{2hd}{vt} = n\lambda_n = \frac{nc}{f_n} \tag{5.5}$$

Solving for time and differentiating with respect to frequency gives:

$$\frac{dt}{df_n} \cong \frac{2hd}{ncv}$$

$$h \cong \frac{dt}{df_n}\left(\frac{ncv}{2d}\right) \tag{5.6}$$

So at times far from CPA, the time-frequency plot of the maxima are linear with a slope which decreases by integer multiples from the steepest slope for n = 1. Measurement of the slope would therefore yield the altitude of the aircraft.

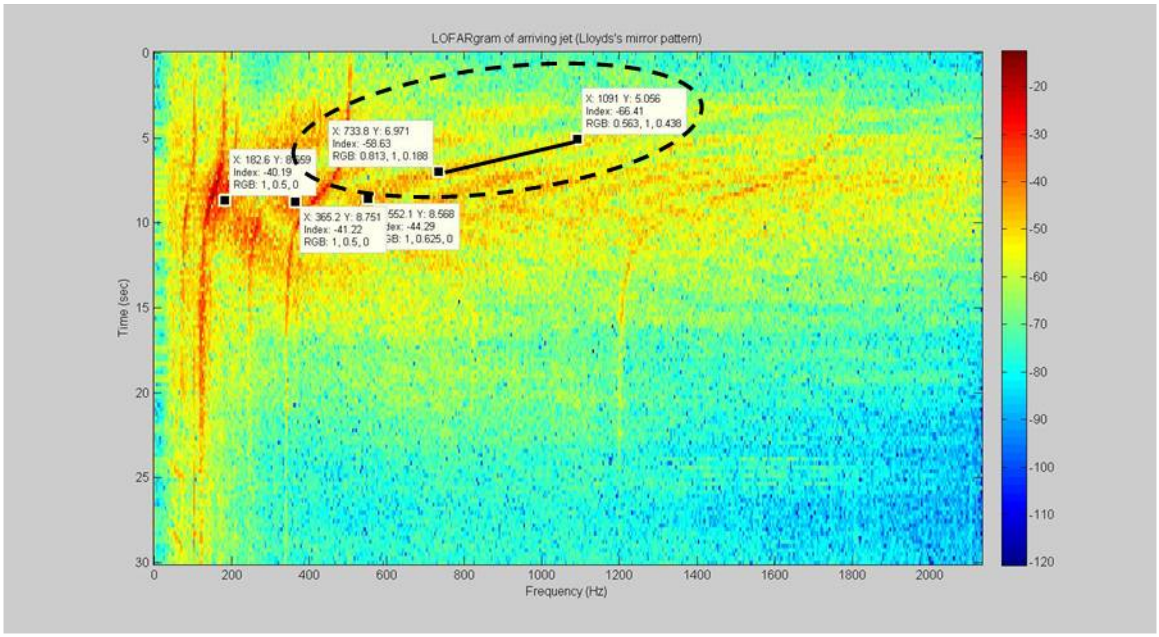

Figure 29. Deriving target altitude from gradient of frequency peaks

Applying this result to the example above, we first determine the n = 1 peak at approximately 182 Hz (Figure 29). This can be checked by verifying the subsequent maxima are indeed integer multiples of this frequency. For example, 2nd maximum is 365 Hz (n = 2.00), 3rd maximum is 552 Hz (n = 3.03).

Next we choose a gradient of frequency peaks. Recall the use of $t \to \infty$ in simplification earlier. It is therefore best to choose the visible frequency gradient that is furthest away from CPA.

In this example, the n = 3 maxima was used. Using d = 0.9m and n = 1,

$$h \cong \frac{6.971 - 5.056}{1091 - 733.8}\left(\frac{(3)(343)(65.2784)}{2(0.9)}\right) = 200\ m$$

c. *Horizontal Distance at CPA*

At CPA, t = 0. (5.2) then simplifies to:

$$\Delta r_{CPA} = \sqrt{(h+d)^2 + R_0^2} - \sqrt{(h-d)^2 + R_0^2} \tag{5.7}$$

This path length difference produces maxima at:

$$f_n = \frac{nc}{\sqrt{(h+d)^2 + R_0^2} - \sqrt{(h-d)^2 + R_0^2}} \tag{5.8}$$

Using the previously estimated altitude h, and the frequencies of the maxima observed at CPA, the exact solution for R_0 can be found using a standard root finding algorithm. If the horizontal range at CPA is small compared to the altitude ($R_o \ll h-d$), as in this case of overflying targets, R_0 can also be estimated explicitly.

$$\Delta r = \sqrt{(h+d)^2 + R_0^2} - \sqrt{(h-d)^2 + R_0^2}$$

$$\Delta r = (h+d)\sqrt{1 + \frac{R_0^2}{(h+d)^2}} - (h-d)\sqrt{1 + \frac{R_0^2}{(h-d)^2}}$$

Assuming $R_o \ll h-d$), this expression can be simplified to:

$$\Delta r \cong (h+d)\left(1 + \frac{1}{2}\frac{R_0^2}{(h+d)^2}\right) - (h-d)\left(1 + \frac{1}{2}\frac{R_0^2}{(h-d)^2}\right) \tag{5.9}$$

Expanding the terms and simplifying gives:

$$\Delta r \cong 2d + \frac{1}{2}\frac{R_0^2}{(h+d)} - \frac{1}{2}\frac{R_0^2}{(h-d)} = 2d + \frac{1}{2}\frac{R_0^2}{h\left(1+\frac{d}{h}\right)} - \frac{1}{2}\frac{R_0^2}{h\left(1-\frac{d}{h}\right)} \qquad (5.10)$$

Now assuming $h >> d$ (200m vs 0.9m in this case), we can further simplify this into:

$$\Delta r \cong 2d + \frac{1}{2}\frac{R_0^2}{h}\left(1-\frac{d}{h}\right) + \frac{1}{2}\frac{R_0^2}{h}\left(1+\frac{d}{h}\right) = 2d - \frac{R_0^2}{h^2}d = d\left(2-\frac{R_0^2}{h^2}\right) \qquad (5.11)$$

From (5.5), $\Delta r \cong \frac{nc}{f_n} \cong d\left(2-\frac{R_0^2}{h^2}\right)$

$$R_o \cong h\sqrt{2-\frac{nc}{f_n d}} \qquad (5.12)$$

Applying to our example, and using the first maximum at 182 Hz, h = 0.95 m (using 0.9 m would result in an imaginary root), d = 200 m,

$$R_o \cong 200\sqrt{2-\frac{(1)(343)}{(182)(0.95)}} \cong 25\,m$$

The exact solution using (5.8) and Matlab 'roots' function gave an answer of 25.6 m, thus validating the prior assumptions.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

The use of a single acoustic vector sensor in tracking commercial aircraft in final landing approaches has been successfully demonstrated. The advantages of a relatively small package with wideband capability could be substantially exploited in mobile systems.

The specific setup used however, did not manage to do so in real-time as desired, which is essentially a function of computing power. Algorithms that are more efficient could potentially improve performance.

The results revealed that vector sensors, since relying heavily on the amplitude and phase information between the particle velocity and pressure channels, are especially sensitive to reflections. The contrast between in-chamber and field testing illustrate the adverse effects of bottom reflections. Curbing such interferences would therefore be an utmost priority in the employment of vector sensors in general. Nonetheless, it was also illustrated that reflections can become a useful source of information when properly analyzed.

In seeking an eventual capability of real-time tracking, continued efforts could be made in various directions:

- Tracking of higher-frequency (500 Hz and above) targets which operate well clear of the minimum working frequency
- Using acoustic-absorbent material designed for frequency band of interest
- Re-positioning sensor to minimize bottom reflections

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. LIST OF EQUIPMENT

Microflown Holland Four Channel Ultimate Sound Probe (USP)—Model UT0901-23/24 (Sensor 323/324)	
Microflown Holland Four Channel Signal Conditioner—Model E0901-23/24	
National Instruments CompactDAQ USB Chassis Model NI 9172	
National Instruments Sound and Vibration DAQ Module—Model NI 9234 (3 total)	
AUSTIN AU-15G Amplifier	
Hewlett Packard Function Generator—Model 33120A	
Kestrel 4000 Handheld Anemometer	
Dell Precision M4500 Laptop Computer (Core i7 1.60 GHz, 4 Gb RAM, 64-bit Windows 7)	-

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. COMPUTER CODE

A. MAIN SCRIPT

main.m

```
%This is the main script to run in real-time, and saves the entire data
%for subsequent processing and analysis.  User to specify:
%Fs - sampling frequency in Hz
%duration - collection duration in seconds

clear all
close all
clc

%Specify parameters
Fs = 4265; %samplng frequency
duration = 5; %seconds

%use acquireData_4ch or acquireData_12ch as appropriate
data = acquireData_4ch(Fs, duration);

%save data to file with system timestamp
dt = datestr(now, 'mmmdd_HHMMSS');
ft = strcat('.\Nfloor\',dt);
save (ft,'data')
```

B. KEY FUNCTIONS

acquireData_4ch.m

```
function [data] = acquireData_4ch (Fs, duration)
%Starts the data acquisition process using Matlab's Data Acquisition
%Toolbox, and via NI DAQmx drivers.

%Inputs from calling function:
%Fs - sampling frequency in Hz
%duration - acquisition duration in seconds

%Variables need to be global so to be used across various functions
global data
global Fs

%Create generic session for NI devices
s = daq.createSession('ni');

%Setting parameters using provided inputs
s.Rate = Fs;
s.DurationInSeconds = duration;

%Add device and channel/s.
%If unsure, type 'daq.getDevices()' to see what's connected and
available
s.addAnalogInputChannel('Dev1',0:3,'voltage');

%The previous steps take a few seconds, and may affect timely
recording.
%The following pause inserts a break, so recording starts almost
%immediately after a key stroke here.
pause()

%Start background listening, and specify function to process data when
%available
lh = s.addlistener('DataAvailable',@plotData);
s.startBackground();
s.wait()
```

acquireData_12ch.m

```
function [data] = acquireData_12ch (Fs, duration)
%Starts the data acquisition process using Matlab's Data Acquisition
%Toolbox, and via NI DAQmx drivers.

%Inputs from calling function:
%Fs - sampling frequency in Hz
%duration - acquisition duration in seconds

%Variables need to be global so to be used across various functions
global data
global Fs

%Create generic session for NI devices
s = daq.createSession('ni');

%Setting parameters using provided inputs
s.Rate = Fs;
s.DurationInSeconds = duration;

%Add device and channel/s.
%If unsure, type 'daq.getDevices()' to see what's connected and
available
s.addAnalogInputChannel('cDAQ1Mod1',0:3,'voltage');
s.addAnalogInputChannel('cDAQ1Mod2',0:3,'voltage');
s.addAnalogInputChannel('cDAQ1Mod3',0:3,'voltage')

%The previous steps take a few seconds, and may affect timely
recording.
%The following pause inserts a break, so recording starts almost
%immediately after a key stroke here.
pause()

%Start background listening, and specify function to process data when
%available
lh = s.addlistener('DataAvailable',@plotData);
s.startBackground();
s.wait()
```

plotData.m

```
function plotData(src,event)
% Called by 'acquireData_4ch.m' or 'acquireData_12ch.m' when data is
% available.  Appends entire data stream to global variable 'data'

    persistent tempData;
    persistent i;

    global data
    global Fs
    if(isempty(tempData))
         tempData = [];

    end

    data = [data;event.Data];


    %Specify data chunk length for subsequent processing
    N = 2^10;
    NFFT = N/2;
    i = floor(length(data)/N);


    j = 0;
    if i > 0 & i > j

        start = (i-1)*N+1
        finish = i*N
        y = data(start:finish,:);

        %Process every 'x' available data chunks to adjust system
response
        %time.  Balance between refresh rate and system lag.
        x = 4;
        if (mod(i,x)==0)
            beamform_4ch(Fs,N,NFFT,y);
        end



        %Plots single-sided amplitude spectrum for feedback in real-
time
        L = length(y);
        NFFT2 = 2^nextpow2(L);
        Y = fft(y,NFFT2)/L;
        f = Fs/2*linspace(0,1,NFFT2/2+1);

        figure(1)
        y2 = 2*abs(Y(1:NFFT2/2+1));
        plot(f,2*abs(Y(1:NFFT2/2+1)))
        axis([0 2000 0 50*mean(y2)]);
        time = num2str(i);
        time = strcat({' '}, time, {' sec'});
        figtitle = strcat('Single-Sided Amplitude Spectrum of y(t)', {'
'}, time);
        title(figtitle)
        xlabel('Frequency (Hz)')
        ylabel('|Y(f)|')


        j = i
    end
end
```

beamforme_4ch.m

```
function beamform_4ch (Fs,binN,N,bindata)
%use tic & toc to time beamformer cycle
tic

global Fpoints targetPosition targetphi targettheta time_taken;


    %DAQ channel assignments
    prs1 = 1; %sensor 324 pressure
    blu1 = 2; %sensor 324 blu velocity Z axis
    grn1 = 3; %sensor 324 grn velocity X axis
    red1 = 4; %sensor 324 red velocity Y axis

    thetainc = 1; %angle increment degrees
    phiinc = 1; %angle increment degrees
    numsens = 1; %total number of sensors
    numel = 4; %number of elements per sensor


    %calculated parameters and constants
    k = linspace(0,N-1,N); %baseline FFT bin numbers
    d = 0.172; %array element spacing meters - measured 172mm ± 2mm
    c = 340; %speed of sound meters/sec @ 20°C sea level
    maxchan = numsens * numel; %total channels
    x = 1; %X axis designator
    y = 2; %Y axis designator
    z = 3; %Z axis designator

    %Vector Sensor Angles relative to Array Apparatus - radians
    thetab1 = 0; %323 blu1 0°
    phib1 = pi/2; %323 blu1 90°
    thetag1 = pi/2; %323 grn1 90°
    phig1 = 0; %323 grn1 0°
    thetar1 = pi/2; %323 red1 90°
    phir1 = pi/2; %323 red1 90°

    %Sensor Element Unit Vectors
    ue(prs1,:) = [0 0 0]; %323 prs1 dummy [0 0 0] - change in udot to
[omni]
    ue(blu1,:) = [sin(thetab1)*cos(phib1) sin(thetab1)*sin(phib1)
cos(thetab1)]; %323 blu1 [0 0 1]
    ue(grn1,:) = [sin(thetag1)*cos(phig1) sin(thetag1)*sin(phig1)
cos(thetag1)]; %323 grn1 [1 0 0]
    ue(red1,:) = [sin(thetar1)*cos(phir1) sin(thetar1)*sin(phir1)
cos(thetar1)]; %323 red1 [0 1 0]

    %Steered Unit Vectors
    thetas = 0:deg2rad(thetainc):pi; %polar steer angle array- radians
    phis = -pi:deg2rad(phiinc):pi; %azimuthal steer angle array upper
hemisphere - radians
    %phis = -pi/2:deg2rad(phiinc):pi/2; %azimuthal steer angle array
upper hemisphere - radians
```

```
    m = length(thetas); %total number of angles in theta direction
    n = length(phis); %total number of angles in psi direction
    p = length(k); %vector length in k direction
    u = sin(thetas') * cos(phis); %steered direction cosine x - m x n
matrix
    v = sin(thetas') * sin(phis); %steereddirection cosine y - m x n
matrix
    w = cos(thetas') * ones(1,n); %steereddirection cosine z - m x n
matrix

    %element unit vector & steer angle dot product array
    %calculate one udot per element, same for all sensors,
    %use first four channels of ue matrix
    %vectorize dot products for faster multiplication in beamformer
    udot = ue(prs1:red1,x)*u(:)' + ue(prs1:red1,y)*v(:)' +
ue(prs1:red1,z)*w(:)'; %1-4 x m*n
    %omni-directional pressure sensors - vector is "1" in all
orientations and aspects
    udot(1,:) = ones(1,m*n); %1 x m*n


    Xc = zeros(maxchan,N);

    %hanning window to reduce sidelobes of signal
    H = hann(binN) * ones(1,maxchan);
    xwin = bindata .* H;
    %clear bindata H

    %convert binary signal data to frequency domain
    Xw = fft(xwin); %by columns binN x maxchan

    %truncate length of signal to match transfer function length N x
maxchan
    X = Xw(1:N,:);
    X = X.'; %change to row data maxchan x N, non conjugate transpose
    clear xwin Xw %save memory


    load '.\txfunction_323.mat'
    Hp1b_BLUE = Hp1B(1:N,:);
    Hp1b_GREEN = Hp1G(1:N,:);
    Hp1b_RED = Hp1R(1:N,:);

    %Apply transfer functions
    Xc(1,:) = X(1,:);
    Xc(2,:) = X(2,:) .* Hp1b_BLUE'; %Hb1=Hp1*Hb1 - pre multiply vel and
prs Xfer funct
    Xc(3,:) = X(3,:) .* Hp1b_GREEN'; %Hg1=Hp1*Hg1 - pre multiply vel
and prs Xfer funct
    Xc(4,:) = X(4,:) .* Hp1b_RED'; %Hr1=Hp1*Hr1 - pre multiply vel and
prs Xfer funct


    %apply band-pass filter
    length(Xc)
    startband = 300;
    stopband = 600;
    startbin = floor(startband*2*N/Fs);
    stopbin = floor(stopband*2*N/Fs);
    Xc2 = zeros(maxchan,N);
    Xc2(:,startbin:stopbin) = Xc(:,startbin:stopbin);

    %Use peakfinder.m to determine source frequency/s
    [peakLoc1, peakMag]=peakfinder(Xc2(1,:));
    peakLoc = peakLoc1.*Fs./(N*2); %convert from bin numbers to actual
frequencies

    %choose maximum peak within earlier specified passband
    [C,I] = max(peakMag);
    Fo = peakLoc(I);
    Fpoints = [Fpoints;Fo]; %store into global variable for appending
onto LOFARgram


    %+-  bins around design freq to truncate - processing bandwidth,
ignore remainder
    Fplot = floor(Fo*(N*2)/Fs) + 1; %k bin for freq Fo
    HBW = Fo/100; %+- freq to offset signal from Fo - broad for live
signals
    halfbandwidth = floor(HBW*(N*2)/Fs) + 1; %+- number of bins to
offset signal
    kshort = k(:,Fplot-halfbandwidth:Fplot+halfbandwidth); %get
relevant k values
    k = kshort; %transfer relevant k values to new k vector
    clear kshort
    p = length(k); %new vector length in k direction


    %select appropriate bins for beamforming
    %Option 1:  Use detected peak frequency Fo +- buffer
    Xcshort = Xc(:,Fplot-halfbandwidth:Fplot+halfbandwidth);

    %Option 2:  Use continuous range of frequencies for noise reduction
average
    %Xcshort = Xc(:,startbin:stopbin);
    Xc = Xcshort;
    clear Xcshort %save memory
    Xc = conj(Xc);


    %%initialize beamformer%%
    %%S is sum of all elements and sensors - the beamformer%%
    S = zeros(m*n,size(Xc,2));  %final beam former output - m*n x p
array

    %Beamformer output
    for q = 1:4;
        %pull element matrix from udot array
        %Theta m * Phi n vector
        udotc = squeeze(udot((mod(q+3,4)+1),:,:)); %m*n x 1
        sumcalc = udotc(:) * Xc(q,:); %m*n x p
        S = S + sumcalc;
    end

    S = sum(abs(S),2);  %sum along k "amplitude", collapse to single
value in k direction
    S = reshape(S, [m n]); %mxn thetaxphi 2D matrix

    %remove SdB points outside 90% max value for clearer phi-theta
plots
    indices = S > max(max(S)).*0.9;
    S = indices.*S;

    %convert to dB
    SdB = 20*log10(abs(S));

    %Find target phi and theta by searching for maximum values
    [SdBmax1,I1] = max(max(SdB,[],1));
    [SdBmax2,I2] = max(max(SdB,[],2));
    phisdeg = rad2deg(phis);
    thetasdeg = rad2deg(thetas);
    targetphi = phisdeg(I1);
    targettheta = thetasdeg(I2);
    targetphirad = phis(I1);
    targetthetarad = thetas(I2);

    %Append onto global vector for final tracking plot
    targetPosition = [targetPosition; [targetphi targettheta]];

    %Convert into coordinate system of 'Cone' subfunction
    x = sin(targetthetarad) * sin(targetphirad) * 20;
    y = cos(targetthetarad) * 20;
    z = sin(targetthetarad) * cos(targetphirad)* 20;

    Cone([0 0 0],[x y z],[0 1],10,'r',0,0,Fo,targettheta,targetphi);


    %Overhead Theta by Phi amplitude plot
    figure (5)
    %clims = [min(min(SdB)) max(max(SdB))];
    %imagesc(rad2deg(phis),rad2deg(thetas),SdB,clims);
    imagesc(rad2deg(phis),rad2deg(thetas),SdB);
    location = strcat({' \phi='}, num2str(targetphi), {' \theta='},
num2str(targettheta));
    figtitle = strcat({'Detected source of '}, num2str(Fo,'%4.0f'), {'
Hz'}, location);
    title(figtitle)
    axis([-180 180 0 180])
    xlabel('\phi (deg)')
    ylabel('\theta (deg)')
    shading interp

    %{
```

```
    %3D rendering of sensor beam pattern.  This allows visual
verification
    of vector sensor's performance, which is has a maximum in the
direction
    of source, and a deep null away from it.

    %convert to cartesian coordinates
    %r = 1;
    xdir = SdB .* u; %x coord normal to yz plane - amplitude
    ydir = SdB .* v; %y coord normal to xz plane - phi
    zdir = SdB .* w; %z coord normal to xy plane - theta

    figure (3)
    axis([0 180 0 180 40 90])
    surf(ydir,xdir,zdir,SdB,'EdgeColor','none'); %nxm 3D plot at freq
of interest
    set(gca,'ZDir','reverse')
    camup([0 1 0]);
    campos([20 15 20])
    title(figtitle)
    xlabel('y axis')
    ylabel('x axis') %phi - n
    zlabel('z axis') %theta - m
    shading interp

    %}

time_taken = [time_taken;toc];
end
```

beamform_12ch.m

```
function beamform_12ch (Fs,binN,N,bindata)
%use tic & toc to time beamformer cycle
tic

global Fpoints targetPosition targetphi targettheta time_taken;


    %DAQ channel assignments
    prs1 = 1; %sensor 324 pressure
    blu1 = 2; %sensor 324 blu velocity Z axis
    grn1 = 3; %sensor 324 grn velocity X axis
    red1 = 4; %sensor 324 red velocity Y axis
    prs2 = 5; %ACO calibrated pressure mic
    blu2 = 6; %ACO calibrated pressure no velocity - dummy
    grn2 = 7; %ACO calibrated pressure no velocity - dummy
    red2 = 8; %ACO calibrated pressure no velocity - dummy
    prs3 = 9; %sensor 323 pressure
    blu3 = 10; %sensor 323 blu velocity Z axis
    grn3 = 11; %sensor 323 grn velocity X axis
    red3 = 12; %sensor 323 red velocity Y axis

    thetainc = 1; %angle increment degrees
    phiinc = 1; %angle increment degrees
    numsens = 3; %total number of sensors
    numel = 4; %number of elements per sensor


    %calculated parameters and constants
    k = linspace(0,N-1,N); %baseline FFT bin numbers
    d = 0.172; %array element spacing meters - measured 172mm ± 2mm
    c = 340; %speed of sound meters/sec @ 20°C sea level
    maxchan = numsens * numel; %number of element channels, 6-8 are
dummys here
    x = 1; %X axis designator
    y = 2; %Y axis designator
    z = 3; %Z axis designator

    %Vector Sensor Angles relative to Array Apparatus - radians
    %323 sensor
    thetab1 = 0; %323 blu1 0°
    phib1 = pi/2; %323 blu1 90°
    thetag1 = pi/2; %323 grn1 90°
    phig1 = 0; %323 grn1 0°
    thetar1 = pi/2; %323 red1 90°
    phir1 = pi/2; %323 red1 90°
    %324 sensor
    thetab3 = 0; %324 blu3 0°
    phib3 = pi/2; %324 blu3 90°
    thetag3 = pi/2; %324 grn3 90°
    phig3 = 0; %324 grn3 0°
    thetar3 = pi/2; %324 red3 90°
    phir3 = pi/2; %324 red3 90°

    %Sensor Element Unit Vectors
    ue(prs1,:) = [0 0 0]; %323 prs1 dummy [0 0 0] - change in udot to
[omni]
    ue(blu1,:) = [sin(thetab1)*cos(phib1) sin(thetab1)*sin(phib1)
cos(thetab1)]; %323 blu1 [0 0 1]
    ue(grn1,:) = [sin(thetag1)*cos(phig1) sin(thetag1)*sin(phig1)
cos(thetag1)]; %323 grn1 [1 0 0]
    ue(red1,:) = [sin(thetar1)*cos(phir1) sin(thetar1)*sin(phir1)
cos(thetar1)]; %323 red1 [0 1 0]
    ue(prs2,:) = [0 0 0]; %aco prs2 dummy [0 0 0] - change in udot to
[omni]
    ue(blu2,:) = [0 0 0]; %aco no vel dummy
    ue(grn2,:) = [0 0 0]; %aco no vel dummy
    ue(red2,:) = [0 0 0]; %aco no vel dummy
    ue(prs3,:) = [0 0 0]; %324 prs3 dummy [0 0 0] - change in udot to
[omni]
    ue(blu3,:) = [sin(thetab3)*cos(phib3) sin(thetab3)*sin(phib3)
cos(thetab3)]; %324 blu3 [0 0 1]
    ue(grn3,:) = [sin(thetag3)*cos(phig3) sin(thetag3)*sin(phig3)
cos(thetag3)]; %324 grn3 [1 0 0]
    ue(red3,:) = [sin(thetar3)*cos(phir3) sin(thetar3)*sin(phir3)
cos(thetar3)]; %324 red3 [0 1 0]

    %Steered Unit Vectors
    thetas = 0:deg2rad(thetainc):pi; %polar steer angle array- radians
    phis = -pi:deg2rad(phiinc):pi; %azimuthal steer angle array upper
hemisphere - radians
    %phis = -pi/2:deg2rad(phiinc):pi/2; %azimuthal steer angle array
upper hemisphere - radians
    m = length(thetas); %total number of angles in theta direction
    n = length(phis); %total number of angles in psi direction
    p = length(k); %vector length in k direction
    u = sin(thetas') * cos(phis); %steered direction cosine x - m x n
matrix
    v = sin(thetas') * sin(phis); %steereddirection cosine y - m x n
matrix
    w = cos(thetas') * ones(1,n); %steereddirection cosine z - m x n
matrix

    %element unit vector & steer angle dot product array
    %calculate one udot per element, same for all sensors,
    %use first four channels of ue matrix
    %vectorize dot products for faster multiplication in beamformer
    udot = ue(prs1:red1,x)*u(:)' + ue(prs1:red1,y)*v(:)' +
ue(prs1:red1,z)*w(:)'; %1-4 x m*n
    %omni-directional pressure sensors - vector is "1" in all
orientations and aspects
    udot(1,:) = ones(1,m*n); %1 x m*n


    Xc = zeros(maxchan,N);

    %hanning window to reduce sidelobes of signal
    H = hann(binN) * ones(1,maxchan);
    xwin = bindata .* H;
    %clear bindata H
    %convert binary signal data to frequency domain
    Xw = fft(xwin); %by columns binN x maxchan

    %truncate length of signal to match transfer function length N x
maxchan
    X = Xw(1:N,:);
    X = X.'; %change to row data maxchan x N, non conjugate transpose
    clear xwin Xw %save memory


    load '.\txfunction_323.mat'
    Hp1b_BLUE = Hp1B(1:N,:);
    Hp1b_GREEN = Hp1G(1:N,:);
    Hp1b_RED = Hp1R(1:N,:);

    %Apply transfer functions
    Xc(1,:) = X(1,:);
    Xc(2,:) = X(2,:) .* Hp1b_BLUE'; %Hb1=Hp1*Hb1 - pre multiply vel and
prs Xfer funct
    Xc(3,:) = X(3,:) .* Hp1b_GREEN'; %Hg1=Hp1*Hg1 - pre multiply vel
and prs Xfer funct
    Xc(4,:) = X(4,:) .* Hp1b_RED'; %Hr1=Hp1*Hr1 - pre multiply vel and
prs Xfer funct

    Xc(9,:) = X(9,:);
    Xc(10,:) = X(10,:) .* Hp1b_BLUE'; %Hb1=Hp1*Hb1 - pre multiply vel
and prs Xfer funct
    Xc(11,:) = X(11,:) .* Hp1b_GREEN'; %Hg1=Hp1*Hg1 - pre multiply vel
and prs Xfer funct
    Xc(12,:) = X(12,:) .* Hp1b_RED'; %Hr1=Hp1*Hr1 - pre multiply vel
and prs Xfer funct



    %apply band-pass filter
    length(Xc)
    startband = 300;
    stopband = 600;
    startbin = floor(startband*2*N/Fs);
    stopbin = floor(stopband*2*N/Fs);
    Xc2 = zeros(maxchan,N);
    Xc2(:,startbin:stopbin) = Xc(:,startbin:stopbin);

    %Use peakfinder.m to determine source frequency/s
    [peakLoc1, peakMag]=peakfinder(Xc2(1,:));
    peakLoc = peakLoc1.*Fs./(N*2); %convert from bin numbers to actual
frequencies

    %choose maximum peak within earlier specified passband
    [C,I] = max(peakMag);
    Fo = peakLoc(I);
    Fpoints = [Fpoints;Fo]; %store into global variable for appending
onto LOFARgram
```

```
    %+-  bins around design freq to truncate - processing bandwidth, 
ignore remainder
    Fplot = floor(Fo*(N*2)/Fs) + 1; %k bin for freq Fo
    HBW = Fo/100; %+- freq to offset signal from Fo - broad for live 
signals
    halfbandwidth = floor(HBW*(N*2)/Fs) + 1; %+- number of bins to 
offset signal
    kshort = k(:,Fplot-halfbandwidth:Fplot+halfbandwidth); %get 
relevant k values
    k = kshort; %transfer relevant k values to new k vector
    clear kshort
    p = length(k); %new vector length in k direction


    %select appropriate bins for beamforming
    %Option 1:  Use detected peak frequency Fo +- buffer
    Xcshort = Xc(:,Fplot-halfbandwidth:Fplot+halfbandwidth);

    %Option 2:  Use continuous range of frequencies for noise reduction 
average
    %Xcshort = Xc(:,startbin:stopbin);
    Xc = Xcshort;
    clear Xcshort %save memory
    Xc = conj(Xc);


    %%initialize beamformer%%
    %%S is sum of all elements and sensors - the beamformer%%
    S = zeros(m*n,size(Xc,2));  %final beam former output - m*n x p 
array

    %Beamformer output
    for q = 9:12;
        %pull element matrix from udot array
        %Theta m * Phi n vector
        udotc = squeeze(udot((mod(q+3,4)+1),:,:)); %m*n x 1
        sumcalc = udotc(:) * Xc(q,:); %m*n x p
        S = S + sumcalc;
    end

    S = sum(abs(S),2);  %sum along k "amplitude", collapse to single 
value in k direction
    S = reshape(S, [m n]); %mxn thetaxphi 2D matrix

    %remove SdB points outside 90% max value for clearer phi-theta 
plots
    indices = S > max(max(S)).*0.9;
    S = indices.*S;

    %convert to dB
    SdB = 20*log10(abs(S));

    %Find target phi and theta by searching for maximum values
    [SdBmax1,I1] = max(max(SdB,[],1));
    [SdBmax2,I2] = max(max(SdB,[],2));
    phisdeg = rad2deg(phis);
    thetasdeg = rad2deg(thetas);
    targetphi = phisdeg(I1);
    targettheta = thetasdeg(I2);
    targetphirad = phis(I1);
    targetthetarad = thetas(I2);

    %Append onto global vector for final tracking plot
    targetPosition = [targetPosition; [targetphi targettheta]];

    %Convert into coordinate system of 'Cone' subfunction
    x = sin(targetthetarad) * sin(targetphirad) * 20;
    y = cos(targetthetarad) * 20;
    z = sin(targetthetarad) * cos(targetphirad)* 20;

    Cone([0 0 0],[x y z],[0 1],10,'r',0,0,Fo,targettheta,targetphi);


    %Overhead Theta by Phi amplitude plot
    figure (5)
    %clims = [min(min(SdB)) max(max(SdB))];
    %imagesc(rad2deg(phis),rad2deg(thetas),SdB,clims);
    imagesc(rad2deg(phis),rad2deg(thetas),SdB);
    location = strcat({' \phi='}, num2str(targetphi), {' \theta='},
num2str(targettheta));
    figtitle = strcat({'Detected source of '}, num2str(Fo,'%4.0f'), {'
Hz'}, location);
    title(figtitle)
    axis([-180 180 0 180])
    xlabel('\phi (deg)')
    ylabel('\theta (deg)')
    shading interp

    %{

    %3D rendering of sensor beam pattern.  This allows visual
verification
    of vector sensor's performance, which is has a maximum in the
direction
    of source, and a deep null away from it.

    %convert to cartesian coordinates
    %r = 1;
    xdir = SdB .* u; %x coord normal to yz plane - amplitude
    ydir = SdB .* v; %y coord normal to xz plane - phi
    zdir = SdB .* w; %z coord normal to xy plane - theta

    figure (3)
    axis([0 180 0 180 40 90])
    surf(ydir,xdir,zdir,SdB,'EdgeColor','none'); %nxm 3D plot at freq
of interest
    set(gca,'ZDir','reverse')
    camup([0 1 0]);
    campos([20 15 20])
    title(figtitle)
    xlabel('y axis')
    ylabel('x axis') %phi - n
    zlabel('z axis') %theta - m
    shading interp

    %}

time_taken = [time_taken;toc];
end
```

C. SUPPORTING FUNCTIONS

peakfinder.m

```
function varargout = peakfinder(x0, sel, thresh, extrema)
% http://www.mathworks.com/matlabcentral/fileexchange/25500
%
% PEAKFINDER Noise tolerant fast peak finding algorithm
%
%   INPUTS:
%       x0 - A real vector from the maxima will be found (required)
%       sel - The amount above surrounding data for a peak to be
%           identified (default = (max(x0)-min(x0))/4). Larger values 
mean
%           the algorithm is more selective in finding peaks.
%       thresh - A threshold value which peaks must be larger than to 
be
%           maxima or smaller than to be minima.
%       extrema - 1 if maxima are desired, -1 if minima are desired
%           (default = maxima, 1)
%   OUTPUTS:
%       peakLoc - The indicies of the identified peaks in x0
%       peakMag - The magnitude of the identified peaks
%
%   [peakLoc] = peakfinder(x0) returns the indicies of local maxima 
that
%       are at least 1/4 the range of the data above surrounding data.
%
%   [peakLoc] = peakfinder(x0,sel) returns the indicies of local maxima
%       that are at least sel above surrounding data.
%
%   [peakLoc] = peakfinder(x0,sel,thresh) returns the indicies of local
%       maxima that are at least sel above surrounding data and larger
%       (smaller) than thresh if you are finding maxima (minima).
%
%   [peakLoc] = peakfinder(x0,sel,thresh,extrema) returns the maxima of 
the
%       data if extrema > 0 and the minima of the data if extrema < 0
%
%   [peakLoc, peakMag] = peakfinder(x0,...) returns the indicies of the
%       local maxima as well as the magnitudes of those maxima
%
%   If called with no output the identified maxima will be plotted 
along
%       with the input data.
%
%   Note: If repeated values are found the first is identified as the 
peak
%
% Ex:
% t = 0:.0001:10;
% x = 12*sin(10*2*pi*t)-3*sin(.1*2*pi*t)+randn(1,numel(t));
% x(1250:1255) = max(x);
% peakfinder(x)
%
% Copyright Nathanael C. Yoder 2011 (nyoder@gmail.com)

% Perform error checking and set defaults if not passed in
error(nargchk(1,4,nargin,'struct'));
error(nargoutchk(0,2,nargout,'struct'));

s = size(x0);
flipData =  s(1) < s(2);
len0 = numel(x0);
if len0 ~= s(1) && len0 ~= s(2)
    error('PEAKFINDER:Input','The input data must be a vector')
elseif isempty(x0)
    varargout = {[],[]};
    return;
end
if ~isreal(x0)
    warning('PEAKFINDER:NotReal','Absolute value of data will be used')
    x0 = abs(x0);
end

if nargin < 2 || isempty(sel)
    sel = (max(x0)-min(x0))/4;
elseif ~isnumeric(sel) || ~isreal(sel)
    sel = (max(x0)-min(x0))/4;
    warning('PEAKFINDER:InvalidSel',...
        'The selectivity must be a real scalar.  A selectivity of %.4g
will be used',sel)
elseif numel(sel) > 1
    warning('PEAKFINDER:InvalidSel',...
        'The selectivity must be a scalar.  The first selectivity value
in the vector will be used.')
    sel = sel(1);
end

if nargin < 3 || isempty(thresh)
    thresh = [];
elseif ~isnumeric(thresh) || ~isreal(thresh)
    thresh = [];
    warning('PEAKFINDER:InvalidThreshold',...
        'The threshold must be a real scalar. No threshold will be
used.')
elseif numel(thresh) > 1
    thresh = thresh(1);
    warning('PEAKFINDER:InvalidThreshold',...
        'The threshold must be a scalar.  The first threshold value in
the vector will be used.')
end

if nargin < 4 || isempty(extrema)
    extrema = 1;
else
    extrema = sign(extrema(1)); % Should only be 1 or -1 but make sure
    if extrema == 0
```

```
        error('PEAKFINDER:ZeroMaxima','Either 1 (for maxima) or -1 (for
minima) must be input for extrema');
    end
end

x0 = extrema*x0(:); % Make it so we are finding maxima regardless
thresh = thresh*extrema; % Adjust threshold according to extrema.
dx0 = diff(x0); % Find derivative
dx0(dx0 == 0) = -eps; % This is so we find the first of repeated values
ind = find(dx0(1:end-1).*dx0(2:end) < 0)+1; % Find where the derivative
changes sign

% Include endpoints in potential peaks and valleys
x = [x0(1);x0(ind);x0(end)];
ind = [1;ind;len0];

% x only has the peaks, valleys, and endpoints
len = numel(x);
minMag = min(x);


if len > 2 % Function with peaks and valleys

    % Set initial parameters for loop
    tempMag = minMag;
    foundPeak = false;
    leftMin = minMag;

    % Deal with first point a little differently since tacked it on
    % Calculate the sign of the derivative since we taked the first
point
    %  on it does not neccessarily alternate like the rest.
    signDx = sign(diff(x(1:3)));
    if signDx(1) <= 0 % The first point is larger or equal to the
second
        ii = 0;
        if signDx(1) == signDx(2) % Want alternating signs
            x(2) = [];
            ind(2) = [];
            len = len-1;
        end
    else % First point is smaller than the second
        ii = 1;
        if signDx(1) == signDx(2) % Want alternating signs
            x(1) = [];
            ind(1) = [];
            len = len-1;
        end
    end

    % Preallocate max number of maxima
    maxPeaks = ceil(len/2);
    peakLoc = zeros(maxPeaks,1);
    peakMag = zeros(maxPeaks,1);
    cInd = 1;
    % Loop through extrema which should be peaks and then valleys
    while ii < len
        ii = ii+1; % This is a peak
        % Reset peak finding if we had a peak and the next peak is
bigger
        %   than the last or the left min was small enough to reset.
        if foundPeak
            tempMag = minMag;
            foundPeak = false;
        end

        % Make sure we don't iterate past the length of our vector
        if ii == len
            break; % We assign the last point differently out of the
loop
        end

        % Found new peak that was lager than temp mag and selectivity
larger
        %   than the minimum to its left.
        if x(ii) > tempMag && x(ii) > leftMin + sel
            tempLoc = ii;
            tempMag = x(ii);
        end

        ii = ii+1; % Move onto the valley
        % Come down at least sel from peak
        if ~foundPeak && tempMag > sel + x(ii)
            foundPeak = true; % We have found a peak
            leftMin = x(ii);
            peakLoc(cInd) = tempLoc; % Add peak to index
            peakMag(cInd) = tempMag;
            cInd = cInd+1;
        elseif x(ii) < leftMin % New left minima
            leftMin = x(ii);
        end
    end

    % Check end point
    if x(end) > tempMag && x(end) > leftMin + sel
        peakLoc(cInd) = len;
        peakMag(cInd) = x(end);
        cInd = cInd + 1;
    elseif ~foundPeak && tempMag > minMag % Check if we still need to
add the last point
        peakLoc(cInd) = tempLoc;
        peakMag(cInd) = tempMag;
        cInd = cInd + 1;
    end

    % Create output
    peakInds = ind(peakLoc(1:cInd-1));
    peakMags = peakMag(1:cInd-1);
else % This is a monotone function where an endpoint is the only peak
    [peakMags,xInd] = max(x);
    if peakMags > minMag + sel
        peakInds = ind(xInd);
    else
        peakMags = [];
        peakInds = [];
    end
end

% Apply threshold value.  Since always finding maxima it will always be
%   larger than the thresh.
if ~isempty(thresh)
    m = peakMags>thresh;
    peakInds = peakInds(m);
    peakMags = peakMags(m);
end



% Rotate data if needed
if flipData
    peakMags = peakMags.';
    peakInds = peakInds.';
end



% Change sign of data if was finding minima
if extrema < 0
    peakMags = -peakMags;
    x0 = -x0;
end
% Plot if no output desired
if nargout == 0
    if isempty(peakInds)
        disp('No significant peaks found')
    else
        figure;
        plot(1:len0,x0,'.-',peakInds,peakMags,'ro','linewidth',2);
    end
else
    varargout = {peakInds,peakMags};
end
```

cone.m

```
function [Cone,EndPlate1,EndPlate2] =
Cone(X1,X2,R,n,cyl_color,closed,lines,Fo,targettheta,targetphi)
%
% This function constructs a cylinder connecting two center points
%
% Usage :
% [Cone,EndPlate1,EndPlate2] = Cone(X1,X2,R,n,cyl_color,closed,lines)
%
% Cone-------Handle of the cone
% EndPlate1------Handle of the Starting End plate
% EndPlate2------Handle of the Ending End plate
% X1 and X2 are the 3x1 vectors of the two points
% R is the radius of the cylinder/cone R(1) = start radius, R(2) = end
radius
% n is the no. of elements on the cylinder circumference (more-->
refined)
% cyl_color is the color definition like 'r','b',[0.52 0.52 0.52]
% closed=1 for closed cylinder or 0 for hollow open cylinder
% lines=1 for displaying the line segments on the cylinder 0 for only
% surface
%
% Typical Inputs
% X1=[10 10 10];
% X2=[35 20 40];
% r=[1 5];
% n=20;
% cyl_color='b';
% closed=1;
%
% NOTE: There is a MATLAB function "cylinder" to revolve a curve about
an
% axis. This "Cylinder" provides more customization like direction and
etc


% Calculating the length of the Cone
length_cyl=norm(X2-X1);

% Creating 2 circles in the YZ plane
t=linspace(0,2*pi,n)';
xa2=R(1)*cos(t);
xa3=R(1)*sin(t);
xb2=R(2)*cos(t);
xb3=R(2)*sin(t);

% Creating the points in the X-Direction
x1=[0 length_cyl];

% Creating (Extruding) the cylinder points in the X-Directions
xx1=repmat(x1,length(xa2),1);
xx2=[xa2 xb2];%xx2=repmat(x2,1,2);
xx3=[xa3 xb3];%xx3=repmat(x3,1,2);
```

```
% Drawing two filled cirlces to close the cylinder
if closed==1
    hold on
    EndPlate1=fill3(xx1(:,1),xx2(:,1),xx3(:,1),'r');
    EndPlate2=fill3(xx1(:,2),xx2(:,2),xx3(:,2),'r');
end

% Plotting the cylinder along the X-Direction with required length
starting
% from Origin
figure(2)
Cone=mesh(xx1,xx2,xx3);
D = 20;
axis([-D D -D D -D D])
%xlabel('y axis')
%ylabel('x axis')
%zlabel('z axis')
camup([0 0 1])
campos([70 15 20])
location = strcat({' \phi='}, num2str(targetphi), {' \theta='},
num2str(targettheta));
figtitle = strcat({'Detected source of '}, num2str(Fo,'%4.0f'), {'
Hz'}, location);
title(figtitle)

% Defining Unit vector along the X-direction
unit_Vx=[1 0 0];

% Calulating the angle between the x direction and the required
direction
% of Cone through dot product
angle_X1X2=acos( dot( unit_Vx,(X2-X1) )/( norm(unit_Vx)*norm(X2-X1))
)*180/pi;

% Finding the axis of rotation (single rotation) to roate the Cone in
% X-direction to the required arbitrary direction through cross product
axis_rot=cross([1 0 0],(X2-X1) );

% Rotating the plotted Cone and the end plate circles to the required
% angles
if angle_X1X2~=0 % Rotation is not needed if required direction is
along X
    rotate(Cone,axis_rot,angle_X1X2,[0 0 0])
    if closed==1
        rotate(EndPlate1,axis_rot,angle_X1X2,[0 0 0])
        rotate(EndPlate2,axis_rot,angle_X1X2,[0 0 0])
    end
end

% Till now Cone has only been aligned with the required direction, but
% position starts from the origin. so it will now be shifted to the
right
% position
if closed==1
    set(EndPlate1,'XData',get(EndPlate1,'XData')+X1(1))
    set(EndPlate1,'YData',get(EndPlate1,'YData')+X1(2))
    set(EndPlate1,'ZData',get(EndPlate1,'ZData')+X1(3))

    set(EndPlate2,'XData',get(EndPlate2,'XData')+X1(1))
    set(EndPlate2,'YData',get(EndPlate2,'YData')+X1(2))
    set(EndPlate2,'ZData',get(EndPlate2,'ZData')+X1(3))
end
set(Cone,'XData',get(Cone,'XData')+X1(1))
set(Cone,'YData',get(Cone,'YData')+X1(2))
set(Cone,'ZData',get(Cone,'ZData')+X1(3))

% Setting the color to the Cone and the end plates
set(Cone,'AmbientStrength',1,'FaceColor',cyl_color,'FaceLighting','gour
aud');%,'EdgeColor','none')
if closed==1
    set([EndPlate1
EndPlate2],'AmbientStrength',1,'FaceColor',cyl_color,'FaceLighting','go
uraud');%,'EdgeColor','none')
else
    EndPlate1=[];
    EndPlate2=[];
end

% If lines are not needed making it disapear
if lines==0
    set(Cone,'EdgeAlpha',0)
end

%shading faceted % faceted flat interp;
%camlight;
%light;
%lighting gouraud; %flat gouraud phong none
material dull; %shiny dull metal
%surcolormap(bone)

%camlight  headlight;
%light('Style','local','Position',[720 0 500]);
light('Style','local','Position',[0 480 500]);
```

D. MISCELLANEOUS

CPA_rootfinder.m

```
%Two methods for solving the horizontal distance at CPA.

n = 1;      %nth frequency maxima
c = 343;    %speed of sound
h = 200;    %target altitude
d = 0.95;   %sensor height
fn = 182;   %frequency

%Exact solution by root-finding.  Take the positive root for obvious
%reasons.
f = @(x)sqrt((h+d).^2 + x.^2)-sqrt((h-d).^2 + x.^2) - n*c/fn;
r = fzero(f,10)

%Approximation when r << h+d
r2 = h*sqrt(2-(n*c)/(fn*d))
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Caulk, Jeffrey V. “Experimental and Theoretical Performance of a Particle Velocity Vector Sensor in a Hybrid Acoustic Beamformer,” Naval Postgraduate School, 2009. Flightaware. (n.d.). [Search term: Monterey Airport]. Retrieved from http://flightaware.com/live/airport/KMRY on 12 Jan 2012.

Hawkes, M. & Nehorai, A. "Acoustic vector-sensor beamforming and capondirection estimation," IEEE Trans. Sig. Proc. 46, pp. 2291–2304, 1998.

Hochwald, B. & Nehorai, A. “Identifiability in array processing models with vectorsensing applications,” IEEE Trans. Sig. Proc. 44, 1996.

Microflown Holland. (n.d.). *Ultimate Sound Probe Datasheet.* [Brochure].Retrieved from http://www.microflown.com/files/media/library/Datasheets/datasheet_uspregular_v10.pdf on 12 Jan 2012.

Microflown Holland. (n.d.). *Ultimate Sound Probe Manual.* [Brochure]. Retrieved from http://www.microflown.com/files/media/library/Manuals/manual_uspregular_v10.pdf on 12 Jan 2012.

Microflown Holland. (n.d.). *Four Channel Signal Conditioner Datasheet.*[Brochure]. Retrieved from http://www.microflown.com/files/media/library/Datasheets/datasheet_mfsc-4_v10.pdf on 12 Jan 2012.

Microflown Holland. (n.d.). *Four Channel Signal Conditioner Manual.*[Brochure]. Retrieved from http://www.microflown.com/files/media/library/Manuals/manual_mfsc-4_v10.pdf on 12 Jan 2012.

Microflown Holland. (n.d.). *Four Channel Signal Conditioner Datasheet.*[Brochure]. Retrieved from http://www.microflown.com/files/media/library/Manuals/manual_mfsc-4_v10.pdf on 12 Jan 2012.

National Instruments. (n.d.). *NI cDAQ-9172 User Guide and Specifications*. [Brochure].Retrieved from http://www.ni.com/pdf/manuals/371747f.pdf on 12 Jan 2012.

National Instruments. (n.d.). *NI USB-9234 NI 9234 Operating Instructions and Specifications*. [Brochure].Retrieved from http://www.ni.com/pdf/manuals/374238c.pdf on 12 Jan 2012.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Daphne Kapolka
   Physics Department
   Naval Postgraduate School
   Monterey, California

4. Kevin Smith
   Physics Department
   Naval Postgraduate School
   Monterey, California

5. Ng Chee Wee
   Singapore Navy